

全国计算机技术与软件专业技术资格（水平）考试指定用书

系统架构设计师教程

杨春晖 主编　孙伟 副主编

全国计算机专业技术资格考试办公室 组编

清华大学出版社

全国计算机技术与软件专业技术资格（水平）考试指定用书

系统架构设计师教程

全国计算机专业技术资格考试办公室　组编

杨春晖　主编　孙伟　副主编

清 华 大 学 出 版 社

北　京

内 容 简 介

本书作为全国计算机技术与软件专业技术资格（水平）考试指定用书，系统地介绍了系统架构设计师的基本要求和应掌握的重点内容。全书共分21章，对计算机网络基础、信息系统基础、系统开发基础、软件架构设计等诸多内容，以及信息安全、系统安全等内容做了全面的阐述。特别是对合格架构师应具备的理论与实践的知识作了详细的讲述。

本书为参加软件水平考试——系统架构设计师考生的必备考试用书。凡通过本考试的考生，便具备了全国认可的，本行业的高级工程师资格。

图书在版编目(CIP)数据

系统架构设计师教程/杨春晖主编. —北京：清华大学出版社，2009.6
（全国计算机技术与软件专业技术资格（水平）考试指定用书）
ISBN 978-7-302-19708-9

I. 系… II. 杨… III. 软件设计-工程技术人员-资格考核-自学参考资料 IV. TP311.5

中国版本图书馆CIP数据核字（2009）第036849号

责任编辑：柴文强 薛 阳
责任校对：徐俊伟
责任印制：
出版发行：清华大学出版社 **地 址：**北京清华大学学研大厦A座
http://www.tup.com.cn **邮 编：**100084
社 总 机：010-62770175 **邮 购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经 销：全国新华书店
开 本：185×230 **印 张：**37 **防伪页：**1 **字 数：**851千字
版 次：2009年6月第1版 **印 次：**2009年6月第1次印刷
印 数：
定 价： 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177转3103 产品编号：031776-01

序　言

软件产业是信息产业的核心之一，是经济社会发展的基础性、先导性和战略性产业，在推进信息化与工业化融合、促进发展方式转变和产业结构升级、维护国家安全等方面有着重要作用。党中央、国务院高度重视软件产业发展，先后出台了 18 号文件、47 号文件等一系列政策措施，营造了良好的发展环境。近年来，我国软件产业进入快速发展期。2007 年销售收入达到 5834 亿元，出口 102.4 亿美元，软件从业人数达 148 万人。全国共认定软件企业超过 1.8 万家，登记备案软件产品超过 5 万个。软件技术创新取得突破，国产操作系统、数据库、中间件等基础软件相继推出并得到了较好的应用。软件与信息服务外包蓬勃发展，软件正版化工作顺利推进。

随着软件产业的快速发展，软件人才需求日益迫切。为适应产业发展需求、规范软件专业人员技术资格，20 余年前全国计算机软件考试创办，率先执行了以考代评政策。近年来，考试作了很多积极的探索，进行了一系列改革，考试名称、考试内容、专业类别、职业岗位也作了相应的变化。目前，考试名称已调整为计算机技术与软件专业技术资格（水平）考试，涉及 5 个专业类别、3 个级别层次共 27 个职业岗位，采取水平考试的形式，执行资格考试政策，并扩展到高级资格，取得了良好效果。20 余年来，累计报考人数近 200 万，影响力不断扩大。程序员、软件设计师、系统分析师、网络工程师、数据库系统工程师的考试标准已与日本相应考试级别实现互认，程序员和软件设计师的考试标准与韩国实现互认。通过考试，一大批软件人才脱颖而出，为加快培育软件人才队伍、推动软件产业健康发展起到了重要作用。

最近，工业和信息化部电子教育与考试中心组织了一批具有较高理论水平和丰富实践经验的专家编写了这套全国计算机技术与软件专业技术资格（水平）考试教材和辅导用书。按照考试大纲的要求，教材和辅导用书全面介绍相关知识与技术，帮助考生学习备考，将为软件考试的规范和完善起到积极作用。

我相信，通过社会各界共同努力，全国计算机技术与软件专业技术资格（水平）考试将更加规范、科学，培养出更多专业技术人才，为加快发展信息产业、推动信息化与工业化融合做出积极贡献。

工业和信息化部副部长 娄勤俭

前　　言

全国计算机技术与软件专业技术资格（水平）考试是国家级“以考代评”的考试，其目的是科学、公正地对全国计算机与软件专业技术人员进行专业资格认定和专业技术水平测试。实施二十年来在社会上产生了很大的影响，对我国软件产业的形成和发展做出了重要的贡献。为适应我国计算机信息技术发展的需求，国家人事部和信息产业部决定将考试的级别拓展到计算机信息技术行业的各个方面，以满足社会上对各类计算机信息技术人才的需要。

系统架构设计师是近年来在国内外迅速成长并发展良好的一个职业，它对系统开发和信息化建设的重要性及给 IT 业所带来的影响不言而喻。在我国，虽然目前该职业在工作内容、工作职责以及工作边界等方面还存在一定的模糊性和不确定性，但它确实是时代发展的需要,并正在实践中不断完善和成熟。

工业和信息化部电子第五研究所受全国计算机专业技术资格考试办公室委托，承担了编写《系统架构设计师教程》的任务，并组织孔学东、杨春晖、李瑜、孙伟等有关专家对本书的框架结构进行了认真研讨，确定与中山大学联合组织本教程的编著工作。本书主要为希望在系统架构领域获得专业水平资格的读者提供必要的信息，还可作为从事或准备从事系统架构设计工作的专业人员的参考书。在考试大纲中，要求考生掌握的知识面很广，每个章节的内容都能构成相关领域基础的一门课程，因此本教程编写的难度很高。考虑到参加该项高级考试的人员已有一定的基础，所以本书只对考试大纲中所涉及到的知识领域的要点加以阐述，限于篇幅不能详细地展开，请读者谅解。

本书编委会主任孔学东，副主任余阳，成员为母春民、杨春晖、孙伟、潘茂林、李瑜。主编杨春晖，副主编孙伟，编写人员包括孔学东、潘茂林、余阳、朱淑华等，编写小组按照《系统架构设计师考试大纲》的要求开展了为期一年的艰苦编著工作，最后由孙伟、杨春晖统稿。编写过程中李瑜、潘勇、杨培亮、刘杰等专家提出了许多宝贵意见，陈平、黄晓坤、冯炳文、范华平、罗雯、唐良俊、刘瑛、杜鹏懿、李晓辉等同志协助做了大量的资料收集、编辑和校对工作，在此深表感谢。

在本书的编写过程中，参考了许多相关的书籍、资料和互联网发布的信息，编者在此对这些参考文献的作者表示感谢。同时感谢清华大学出版社在本书出版过程中所给予的支持和帮助。

因水平有限，书中难免存在错漏和不妥之处，望读者指正，以利改进和提高。

编　者
2008 年于广州

目　　录

第 1 章　绪　　论

随着技术的进步，信息系统的规模越来越大，复杂程度越来越高，系统的结构显得越来越重要。对于大规模的复杂系统来说，对总体的系统结构设计比起对计算的算法和数据结构的选择已经变得更重要，在这种情况下，人们认识到系统架构的重要性，设计并确定系统整体结构的质量成为了重要的议题。系统架构对于系统开发时所涉及到的成熟产品与相关的组织整合问题具有非常重要的作用，而系统架构师正是解决这些问题的专家。系统架构作为集成技术框架规范了开发和实现系统所必需的技术层面的互动，作为开发内容框架影响了开发组织和个人的互动，因此，技术和组织因素也是系统架构要讨论的主要话题。在系统开发项目中，系统架构师是项目的总设计师，是生产企业新产品、新技术体系的构建者，是目前系统开发中急需的高层次技术人才。

系统架构师是近几年来在国内外迅速成长并发展良好的一个职业，它对系统开发和信息化建设的重要性及给 IT 业所带来的影响是不言而喻的。在我国，虽然系统架构师的职业在工作内容、工作职责以及工作边界等方面还存在一定的模糊性和不确定性，但它确实是时代发展的需要，并正在实践中不断完善和成熟。

1.1　系统架构的概念及其发展历史

1.1.1　系统架构的概念

架构是一个古老的研究领域。在现实中，很多人认为架构就是一个有关建造一个物理结构的学科。但是，在设计一个信息系统架构时，计划（规划）的概念把架构和建设分割开来。

古代的文明对架构的发展有三个主要的贡献。一是多个建筑结构的完美结合。例如在用于装饰和建筑元素中的横梁、拱扇和柱子间的结合。第二个是建筑装饰形式和模式的广泛普及，这当中许多已经成为东西方宝贵文化遗产的一部分，许多还在被用于今天的建筑当中。第三就是有序规划的概念，规划是架构的基石，也就是这三个贡献中最重要的。现代信息系统的“架构”要素亦继承了这三个要素，即构件、模式和规划。

现代信息系统的“架构”本质上存在两个层次：一个是概念的层次，一个是物理的层次。而概念层次则包含了艺术、科学、方法和建设风格。物理的层次是指在一系列的架构工作之后而产生的物理结构及其相互作用的结果。

在实际工作中，为了有效地管理公司和运营业务，首先必须定义和建立一系列清晰

的、实用的信息及其处理流程。这就是在一个企业中的企业总体业务架构观念，所谓软件架构必须支持这一观念。

目前，软件架构已经成为软件工程领域的研究热点。作为大型软件系统与软件产品线开发中的关键技术之一，已发展为软件工程领域的一个独立学科分支。由于所属的专业领域、学术研究和实践内容的不同，研究人员对软件架构有不同的理解和定义。这里，定义如下：

软件系统架构是关于软件系统的结构、行为和属性的高级抽象。在描述阶段，其对象是直接构成系统的抽象组件以及各个组件之间的连接规则，特别是相对细致地描述组件之间的通讯。在实现阶段，这些抽象组件被细化为实际的组件，比如具体类或者对象。软件系统架构不仅指定了软件系统的组织结构和拓扑结构，而且显示了系统需求和构成组件之间的对应关系，包括设计决策的基本方法和基本原理。

1.1.2 简要的发展历史

企业软件架构（Enterprise Software Architecture），也叫做企业架构，是应用全面的和严格的方法描述一个针对信息系统、流程处理、个人和组织当前和/或未来行为的抽象结构集合，所以它们与组织的核心目标和战略方向结合，尽管一般来说与信息技术高度相关，但也与商业流程优化密切相关，因此也涉及商业模式、功能管理和过程架构。

企业软件架构的雏形来自企业建模的理论和思想。在 20 世纪 80 年代早期，除了学术界，很少有人对企业流程再造或企业建模的思想感兴趣，而且使用的理论和模型通常被限于某个信息系统的设计和开发。

到 20 世纪 80 年代中期，还在 IBM 工作的 John Zachman 首先引入“信息系统架构框架”的概念。Zachman 被公认为是企业架构领域的开拓者，他认为使用一个逻辑的企业构造蓝图（即一个架构）来定义和控制企业系统和其组件的集成是非常有用的。为此，Zachman 提出从信息、流程、网络、人员、时间和基本原理等 6 个视角来分析企业，并提供了与这些视角相对应的 6 个模型，包括语义、概念、逻辑、物理、组件和功能模型。

当时，Zachman 并没有使用“企业架构”的概念。1996 年美国的 Clinger-Cohen 法案（以前被称作信息技术管理改革法案）导致了术语“IT 架构”的产生。这部法案的主旨是，美国政府指导下属联邦政府机构通过建立综合方法来管理信息技术的引入、使用和处置等。Clinger-Cohen 法案要求政府机构的（Chief Information Officer，CIO）要负责开发、维护和帮助一个合理的和集成的 IT 架构（IT Architecture，ITA）的实施，当时的术语 ITA，现在被解释为 IT 企业架构。

因此，企业软件架构的最早应用是在一些美国的政府机构，美国政府对企业架构应用的推动也发挥了十分重要的作用。自从 Zachman 框架引入后，首先是美国国家技术标准研究所在 1989 年发布 NIST 框架，从此联邦政府内出现了许多框架，其他联邦实体也发布了企业架构框架，包括国防部和财政部等。

1999 年 9 月，美国联邦 CIO 委员会出版了联邦企业架构框架，它的意图是为联邦机构提供一个架构的公共结构，以利于这些联邦机构间的公共业务流程、技术引入、信息流和系统投资的协调等。

联邦企业架构框架定义了一个 IT 企业架构作为战略信息资产库，它定义了业务、运营业务所必须的业务信息，支持业务运行的必要的 IT 技术，响应业务变革实施新技术所必须的变革流程等要素。

随后，美国的管理和预算办公室（OMB）发布的 OMB Circular A-130，要求机构记录和提交他们的初始的企业架构到 OMB，并对架构发生的重大变革进行更新。这给了 OMB 一个责任，即帮助推动政府机构内和政府机构间的企业架构的开发，并支持通过使用 IT 来改进政府运营能力。

2002 年 2 月，OMB 建立了一个联邦企业架构程序管理办公室来开发 FEA，它的作用是，在联邦机构程序内和跨机构程序间，通过跨部门的分析来找到重复的投资，找到相互的差距，有助于在联邦政府范围内的协作、互操作和交互作用。

企业软件架构的理念很快就得到咨询公司和研究机构认可，被 Gartner 收购的 META Group 是最早对企业软件架构进行分析和研究的主要咨询公司。2000 年，META Group 发布《企业体系机构桌面参考》，提供了一个经验证的实施企业软件架构的方法论，意图成功地构建业务战略和技术实施之间的桥梁。在咨询和研究机构带动下，IBM、微软、HP、EDS 等 IT 厂商也纷纷把目光集聚到了企业软件架构，希望能够从企业这个视角来定位其产品和服务。

随后，政府、应用企业、咨询和研究机构、厂商广泛参与，企业架构标准化的工作越来越重要，也产生了一些研究团体和标准框架。目前，业界最有名的企业架构框架是 TOGAF（The Open Group Architecture Framework，Open Group 架构框架），TOGAF 是一个行业标准的架构框架，它可以被任何希望开发一个信息系统架构的组织在组织内免费使用。

从 20 世纪 90 年代中期开始，TOGAF 已经被一些世界领先的 IT 客户和厂商开发和持续演进。与 TOGAF 类似的架构包括联邦政府企业架构框架、联邦政府企业架构指南、财政部企业架构框架、Spewak 的企业架构规划、Zachman 框架、OMG 的 MOD 等。

企业软件架构实施的主体是企业，企业的需求才是软件架构发展的引擎。而企业软件的需求来源广泛，企业信息化需要支持市场需求、环境要求、经营需要、技术发展、用户要求以及法律需求，涉及企业的各个业务领域，而几乎所有领域都能够和信息技术相结合构成企业信息化项目。

软件架构的研究已发展为软件工程领域的一个独立学科分支，研究主要包括软件架构描述语言、软件架构的描述与表示、软件架构的分析与验证、基于架构的软件维护与演化、软件架构的可靠性等方面。

1.2 系统架构师的定义与职业素质

通常从组织上划分，架构师分为以下几大类：业务架构师（Business Architect）、主题领域架构师（Domain Architect）、技术架构师（Technology Architect）、项目架构师（J2EE架构师、.NET架构师等）以及我们本书所阐述的系统架构师（System Architecture）。

1.2.1 系统架构师的定义

系统架构师是系统或产品线的设计责任人，是一个负责理解和管理并最终确认和评估非功能性系统需求（如软件的可维护性、性能、复用性、可靠性、有效性和可测试性等），给出开发规范，搭建系统实现的核心构架，对整个软件架构、关键构件、接口进行总体设计并澄清关键技术细节的高级技术人员。

系统架构师主要着眼于系统的“技术实现”，同时还要考虑系统的“组织协调”。因此，系统架构师是特定的开发平台、语言、工具的大师，对常见应用场景能及时给出最恰当的解决方案，同时要对所属的开发团队有足够的了解，能够评估该开发团队实现特定的功能需求目标的资源代价。可以说，系统架构师是信息系统开发和演进的全方位技术与管理人才。

1.2.2 系统架构师技术素质

系统架构师通常负责公司系统的架构设计与持续改进，承担从业务向技术系统转换的桥梁作用；协助项目经理制定项目计划和控制项目进度；需要承担技术管理工作，如负责组织技术研究和攻关，负责组织和管理技术培训工作，管理技术支撑团队并给项目、产品开发实施团队提供技术保障。因此一个好的系统架构师的技术素质十分重要，通常系统架构师需要具有系统思维的能力，还必须具备以下技术素质：

- 具备丰富的一线大中型开发项目的总体规划、方案设计及技术队伍管理经验。
- 具备软件行业工作经验，熟悉业务领域的技术应用和发展。
- 具有项目管理理论基础，并在应用系统开发平台和项目管理上有实践经验。
- 对相关的技术标准有深刻的认识，对软件工程标准规范有良好的把握。
- 具备C/S或B/S体系结构或特定领域软件产品开发及架构和设计的经验。
- 具有面向对象分析（object-Oriented Analysis，OOA）、设计（Object-Oriented Design，OOD）、开发（Object-Oriented Programming，OOP）能力，精通UML和XML等，熟练使用Rational Rose、PowerDesigner等CASE工具进行设计开发。
- 对相关编程技术（如PHP/.Net/JAVA）及整个解决方案有深刻的理解及熟练的应用，并且精通架构和设计模式（如WebService/J2EE），并在此基础上设计产品

框架。

- 精通大型数据库如 Oracle、Sql Server、MySQL 等的开发。
- 对计算机系统、网络和安全、应用系统架构等有全面的认识。
- 良好的团队意识和协作精神，有较强的内外沟通能力。

1.2.3　系统架构师管理素质

系统架构师管理素质是必须强调的。它包括远见、诚信、果断的领导素质。系统架构师是一个高效工作团队的建造者。作为核心的高层技术管理人员，架构师必须尽可能使所有团队人员的想法保持一致，为一个项目制订一个清晰的、强制性的、有远见的目标作为整个团队的动力，从而为达到整体目标所作的权衡提供基础。作为技术的领导者，系统架构师必须提供特定的方法和模型作为理想的技术解决方案；并排除各种非系统相关因素的影响。作为一个技术管理者，系统架构师在需要做出决定时，必须避免犹豫，必须具备及时解决技术问题的紧迫感和自信心。

1.2.4　系统架构师与其他团队角色的协调

关于系统架构师的定位，在很多资料中都没有明确的表述，这里可以参照系统开发中的主要角色的给出描述。

根据全国计算机技术与软件专业技术资格（水平）考试的安排，其中作为高级工程师级别的职位有项目管理师、系统分析师和系统架构师（这里的系统架构主要是指软件系统的架构）。考试大纲对这三个职位的要求和职责定义如下。

（1）项目管理师：掌握信息系统项目管理的知识体系，具备管理大型、复杂信息系统项目和多项目的经验和能力；能根据需求组织制定可行的项目管理计划；能够组织项目实施，对项目的人员、资金、设备、进度和质量等进行管理，并能根据实际情况及时做出调整，系统地监督项目实施过程的绩效，保证项目在一定的约束条件下到达既定的项目目标；能分析和评估项目管理计划和成果；能在项目管理进展的早期发现问题，并有预防问题的措施；能协调项目所涉及的相关人员。即项目管理师的主要职责是负责整个项目的实施和控制，协调各种资源（包括组织内部资源和客户资源）。

（2）系统分析师：熟悉应用领域的业务，能分析用户的需求和约束条件，写出信息系统需求规格说明书，制订项目开发计划，协调项目开发与运行所涉及的各类人员；能指导制订企业的战略数据规划，组织开发项目；能评估和选用适宜的开发方法和工具；能按照标准规范编写系统分析、设计文档；能对开发过程进行质量控制与进度控制；能具体指导项目开发。即系统分析师的主要职责是获取并分析用户的需求，形成规范化的文档，指导整个项目的开发，需要与客户不断的交流，熟悉应用领域的业务。

（3）系统架构师：能够根据用户需求，结合用户应用领域的实际情况，设计正确、合理的软件构架，维护系统构件及其接口，并确保系统构架具有良好的性能；能够对项

目进行系统构架级的描述、分析、设计与评估；能够按照相关标准编写相应的设计文档；具有扎实的理论功底、广博的知识面，能够与系统分析师、项目管理师相互协作、配合工作。即系统架构师的职责更加强调整体的、宏观的系统设计，重点在架构级别上。重点要对架构进行描述、分析和评估。

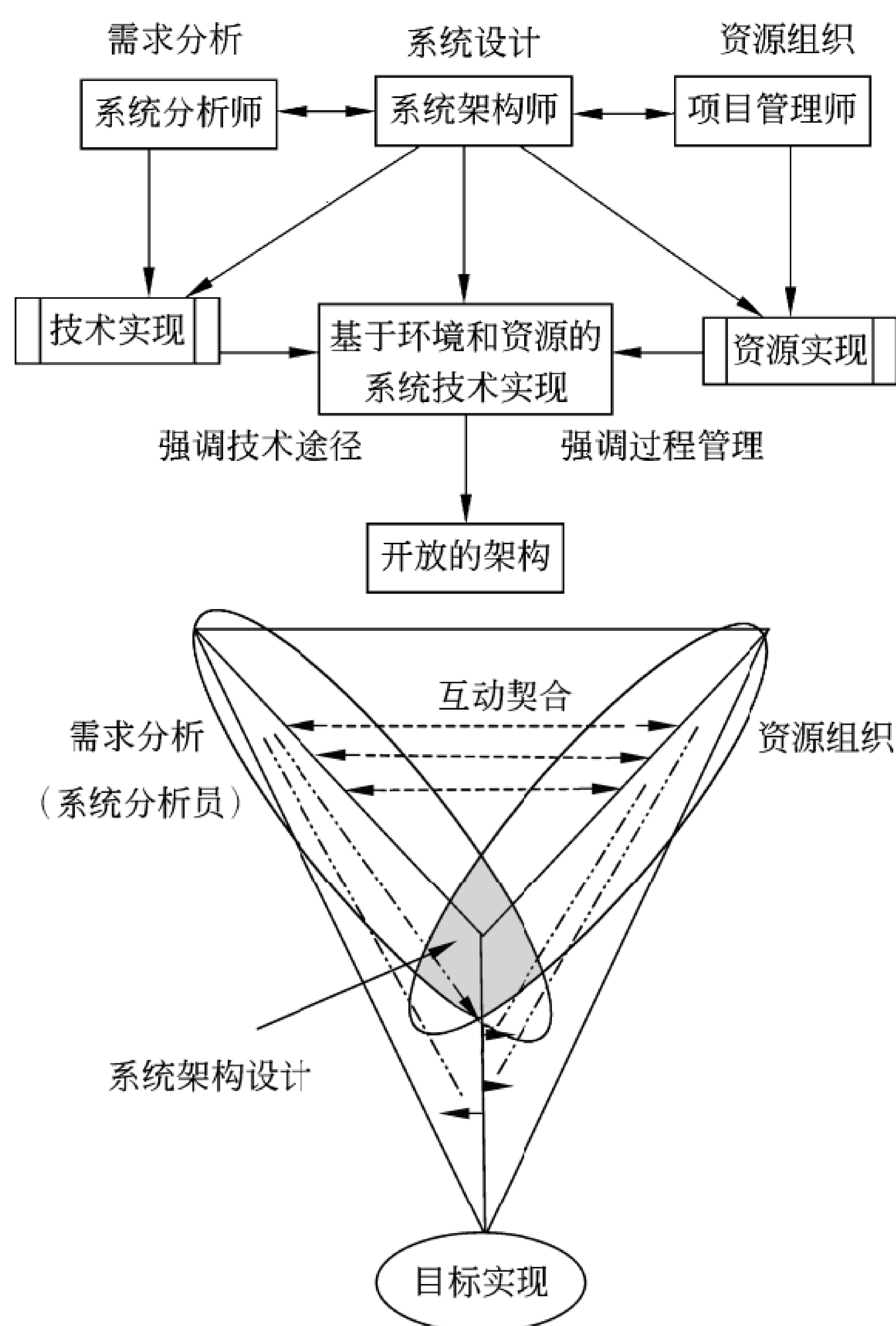


图 1-1　系统架构师的定位

图 1-1 的上图反映了传统的系统开发：通过对用户的需求分析，组织必要的资源和设施，选择设计合适的架构，然后由项目经理统筹安排组织实施（包括人、财和物），这是一个反复互动，逐步契合的过程。

由上面所述可以看出，在传统的系统开发中，系统开发进度及统筹的管理主要由项目经理来完成，需求分析及技术实现主要由系统分析员或设计员来完成。由于项目经理、系统分析员或者程序员从各自职位角度出发产生冲突的制约，不可能很好地给出开发规范，搭建系统实现的核心构架，并澄清技术细节、扫清主要难点的技术，或者说最终难以确认和评估技术对需求实现产生的影响。所以我们把系统架构设计师定位在图 1-1 的下图中两个椭圆相交的部分，他负责整个系统的战略策划和技术实现。图 1-1 的上图说

明架构设计先行和系统架构师、系统分析师、项目管理师三者的相互关系与作用。

1．系统构架师与项目经理的关系及区别

软件项目经理是指对项目控制/管理，关注项目本身的进度、质量、分配、调动、协调，管理好人、财、物等资源的负责人。对于软件项目经理来讲，职责包括项目计划、进度跟踪/监控、质量保证、配置/发布/版本/变更管理、人员绩效评估等方面。优秀的项目经理需要的素质，并不仅在于会使用几种软件或是了解若干抽象的方法论原则，更重要的在于从大量项目实践中获得的宝贵经验，以及交流、协调、激励的能力，甚至还应具备某种个性魅力或领袖气质（charisma）。一般来讲，技术人员重技术而倾向于忽视“人”的因素，而这正是项目经理管理活动的一个主要方面。项目经理还必须能够应付开发过程中大量的偶发事件和杂务。

在一个项目中，推动项目技术发展的是系统构架师。在技术方面，项目经理（项目管理师）配合系统构架师并提供各个方面的支持，其主要职责是与内外部沟通和管理资源（包括人），系统构架师则要负责提出系统的总体构架，并给出开发指导。

2．系统构架师与系统分析师的关系及区别

系统分析师（system analyst）是在系统开发中进行业务需求分析、系统需求分析、可行性分析、业务建模和指导项目开发的人。其工作特点是与行业专家、用户沟通，及时与项目经理（项目管理师）、软件架构师协商，分析项目具备的特点、成本、风险等，考虑实现的模型。系统分析师所面临的往往是有许多不确定性的事件，需要对这些不确定的事件进行分析、总结，使之得出一个相对可靠的确定性结论或实施方案模型。一般意义上讲，系统分析师的水平将影响系统开发的质量，甚至成败。在一个完善的系统开发队伍中，一般应有项目管理师、软件架构师、系统分析师、软件设计师、测试工程师、数据库工程师、程序员和质量保证人员等不同的职位，还需要有业务专家和其他辅助人员。对于大型企业或项目，如果一人承担多个角色，往往容易发生顾此失彼的现象。

系统分析师对业务系统进行分析、建模，他的任务、目标是明确的。系统架构师协同系统分析师的工作，建议系统分析师按什么标准，什么工具，什么模式，什么技术去思考系统。同时，系统架构师应该对系统分析师所提出的问题，碰到的难题及时地提出解决的方法。

1.3　系统架构师知识结构

软件系统架构师综合的知识能力结构包括 9 个方面，即：

（1）战略规划能力。

（2）业务流程建模能力。

（3）信息数据架构能力。

（4）技术架构选择和实现能力。

（5）应用系统架构的解决和实现能力。

（6）基础 IT 知识及基础设施、资源调配的能力。

（7）信息安全技术支持与管理保障能力。

（8）IT 审计、治理与基本需求分析、获取能力。

（9）面向软件系统可靠性与系统生命周期的质量保障服务能力。

作为系统架构师，必须成为所在开发团队的技术路线引导者；具有很强的系统思维的能力；需要从大量互相冲突的系统方法和工具中区分出哪些是有效的，哪些是无效的。架构师应当是一个成熟的、丰富的、有经验的、有良好教育的、学习快捷、善沟通和决策能力强的人。丰富是指他必须具有业务领域方面的工作知识，知识来源于经验或者教育。他必须广泛了解各种技术并精通一种特定技术，至少了解计算机通用技术以便确定哪种技术最优，或组织团队开展技术评估。优秀的架构师能考虑并评估所有可用来解决问题的总体技术方案。需要良好的书面和口头沟通技巧，一般通过可视化模型和小组讨论来沟通指导团队确保开发人员按照架构建造系统。

因此，系统架构师知识维度可以总结为“多层次+多方面”。所谓多层次，意味着系统架构师必须在体系结构、计算机软硬件与网络基础知识、信息化基础知识、信息安全与可靠性基础知识等基本功的层面上受过良好的教育和快捷的学习能力；还须在系统架构设计方法、设计模式、设计流程以及各种模型等方面有丰富的经验，广泛了解各种构件产品和技术并精通一种特定领域的架构设计；进一步，还须在系统架构设计实践层面，有自己的认识和理解，同时具有很强的表述能力；所谓多方面，意味着系统架构师在每个知识层面上必须具有技术、管理、心理和艺术等多方面的知识和能力。这和系统架构师的多角色特点是相关的。本书也正是从这个角度来介绍系统架构的知识体系，即从系统构件、模式和规划三个方面的技术基础、原理和方法的角度编写而成的关于软件架构师的基本知识结构和水平的教材。

1.4 从开发人员到架构师

软件架构师一般都是具备计算机科学或软件工程的知识，由程序员做起，然后再慢慢成长为架构师的。在国内，很多大学目前还没有设立软件架构的学位课程，随着 IT 业界对设计和架构的兴趣日渐高涨，在学校课程中增加部分相关内容已不能满足产业发展的需要。一方面，大学要加强软件架构学课程的建设，另一方面，软件架构师的成长还应该有一个实践的教育过程，并不是简单地通过学校的理论学习或者通过某软件公司的认证就能成为合格的软件架构师。除了在学校学习信息系统综合知识外，软件架构师的大部分知识和经验将来自实际开发工作。根据软件架构师的任职条件，一名合格的软件架构师的成长应该经历 8 年以上的软件项目开发实际工作经验。一般需要经历程序员、软件设计师等阶段，然后再逐步成长为软件架构师。

当然，并不是每一位程序员经过 8 年后都可以成长为软件架构师。一个软件工程师在充分掌握了软件架构师工作所必需的基本理论和技能后，如何得到和利用机会、如何利用所掌握的技能进行应用系统的合理架构、如何不断的抽象和总结自己的架构模式、如何深入行业总结规律，成为能够胜任分析、架构为一体的精英人才，这是机遇、努力和天赋的综合结果。

就目前来看，国内软件架构师的培养途径主要有两种方式：一种是大学（软件学院）教育方式，另一种是个人自我培养然后再进行相应的培训和认证。但是，不管哪种方式都有其不足之处。

软件学院的培养方式能够系统的学习软件架构师必需的知识体系，但是，软件架构师不是简单的通过理论学习就能够培养出来的，软件学院的学生可能缺乏必要的设计、开发经验和相关的领域知识。尽管软件学院也强调给予学生实践的机会，但毕竟这种机会是有限的。即使有充分的机会，也没有足够的时间在实践中获得广泛的检验和验证。也包含一些管理因素，如有关“三分之一的师资来自企业”的规定，在部分软件学院中也没有得到真正落实，导致传授给学生的还是一些纯理论知识。

自我培养方式的主要对象是具有一定年限的软件开发和设计人员，如Microsoft、IBM、Sun 等公司的软件架构师认证对学员的基础并没有具体的要求，只要交纳规定的费用，然后进行几天的集中培训，通过考试就发给学员证书，甚至不需要考试就直接发放证书。这些开发人员在自我培养的过程中不一定能够系统的学习软件架构师的理论知识，他们只具有一定的开发和设计经验，仅仅经过几天的培训，是不太可能培养出合格的软件架构师的。而且，作为某个厂商的培训和认证，其最终目的是培育自己的市场，培养一批忠诚的用户或产品的代言人，而不是为中国培养软件架构师。

在国外，软件架构师的培养与认证具有严格的过程，明确规定了教育目标、认证的要求和学习课程等方面的内容。下面，介绍三个组织的软件架构师的认证情况。

1．UC Irvine

在 UC Irvine 的软件架构师认证计划中，为了拿到软件架构师 C 级认证，学员必须完成 11 个单元的必修课程和至少 4 个单元的选修课程。这些课程如下：

- 必修课程：软件系统建模和分析概论（2 个单元）、系统分析基础（3 个单元）、用户需求的分析和文档化（3 个单元）、软件架构项目（3 个单元）。
- 选修课程：信息系统项目管理（2 个单元）、系统性能建模（2.5 个单元）、管理业务改进项目（2.5 个单元）。

UC Irvine 的软件架构师认证要求学员具有业务系统建模，决定用户需求，评价业务过程的能力，掌握项目管理技术，能设计完善的、具有最佳可适应性和可扩展性的架构。该认证程序以一门实践课程结束，在实践课程中，学员从头开始，设计一个大规模软件解决方案的架构。

2．CMU/SEI

SEI 在软件架构师方面的认证包括三个职位，分别是软件架构师、ATAM 评估师和 ATAM 主任评估师。这些认证都需要学习两年的课程。其中软件架构师需要学习的课程有：软件架构原理与实践、软件架构文档化、软件架构设计与分析和软件产品线。

3．iCMG

iCMG 对软件架构师的认证强调 7 个层次的课程学习。

软件架构师作为软件的总设计师，其水平和能力直接决定了软件系统的总体性能，对软件架构师的认证是十分重要和紧迫的。全国计算机技术与软件专业技术资格（水平）考试设立系统架构设计师级别的认证考试，是解决软件架构师认证问题的重要途径。

第 2 章　计算机与网络基础知识

计算机系统由硬件和软件两部分组成。计算机系统中的软件通常分为系统软件和应用软件两大类。

系统软件支持应用软件的运行，为用户开发应用软件提供平台，用户可以使用它，但不能随意修改它。常用的系统软件有操作系统、语言处理程序、连接程序、诊断程序和数据库管理系统等。

应用软件是指计算机用户利用计算机的软、硬件资源为某一专门的应用目的而开发的软件。例如：科学计算、工程设计、数据处理、事务处理、过程控制等方面的程序，以及文字处理、表格处理、辅助设计（CAD）和实时处理等软件。

2.1　操作系统基础知识

操作系统（Operating System，OS）是计算机系统的核心系统软件，其他软件是建立在操作系统基础上的，并在操作系统的统一管理和支持下运行。操作系统与计算机系统软/硬件的关系如图 2-1 所示。

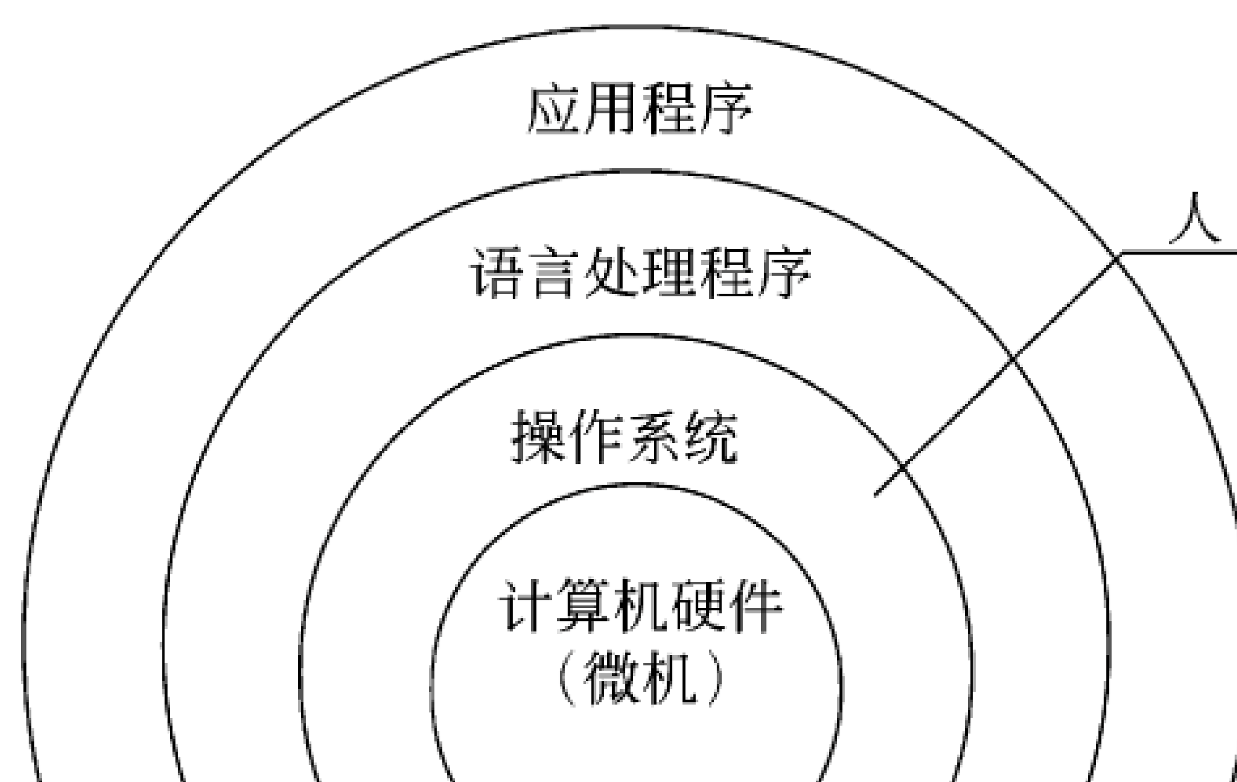


图 2-1　操作系统与计算机系统软/硬件的关系

2.1.1　操作系统的原理、类型和结构

1．操作系统定义

计算机系统的硬件资源包括中央处理机（CPU）、存储器（主存与外存）和输入/输出设备等物理设备。计算机系统的软件资源是以文件形式保存在存储器上的程序和数据等信息。操作系统既有效地组织和管理系统中的各种软、硬件资源，合理地组织计算机系统的工作流程，又控制程序的执行，并且为用户使用计算机提供了一个良好的环境和

友好的接口，从而使用户能充分利用计算机资源，提高系统的效率。

操作系统的作用如下。

（1）通过资源管理，提高计算机系统的效率。

（2）改善人机界面，向用户提供友好的工作环境。

2．操作系统分类

操作系统按功能不同可分为：单用户操作系统和批处理操作系统；分时操作系统和实时操作系统；网络操作系统和分布式操作系统；以及嵌入式操作系统。

3．操作系统的特征

操作系统的 4 个特征是并发性（concurrency）、共享性（sharing）、虚拟性（virtual）和不确定性（non-determinacy）。

4．操作系统的功能

操作系统的五大管理功能是进程管理、文件管理、存储管理、设备管理和作业管理。

2.1.2 处理机与进程管理

进程（process）是资源分配和独立运行的基本单位。研究操作系统的进程，实质上是研究系统中诸进程之间的并发特性以及进程之间的相互制约性。

1．进程的定义及其分类

进程是程序的一次执行，该程序可以和其他程序并发执行。进程通常由程序、数据及进程控制块（Process Control Block，PCB）组成。PCB 描述了进程的基本情况，是进程存在的唯一标志。

程序和进程的区别为程序是静态的指令序列，进程是为执行该程序的线程而保留的资源集。

进程依性质不同可分为：系统进程和用户进程；父进程和子进程。

2．进程的状态转换与控制

进程一般有 2 种基本状态：就绪、运行和阻塞。如图 2-2（a）所示为进程基本状态及其转换，也称三态模型。

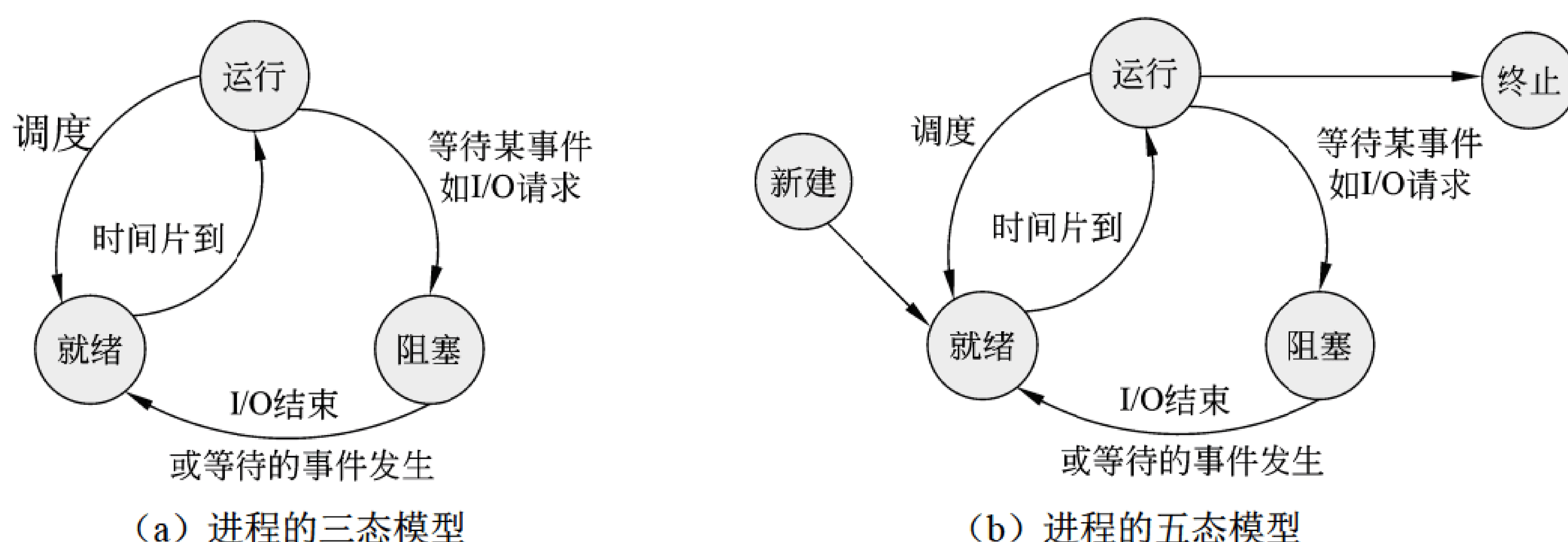


图 2-2 进程基本状态及其转换

进程的五态模型引入了新建态和终止态，如图 2-2（b）所示。具有挂起状态的进程状态及其转换，如图 2-3 所示。

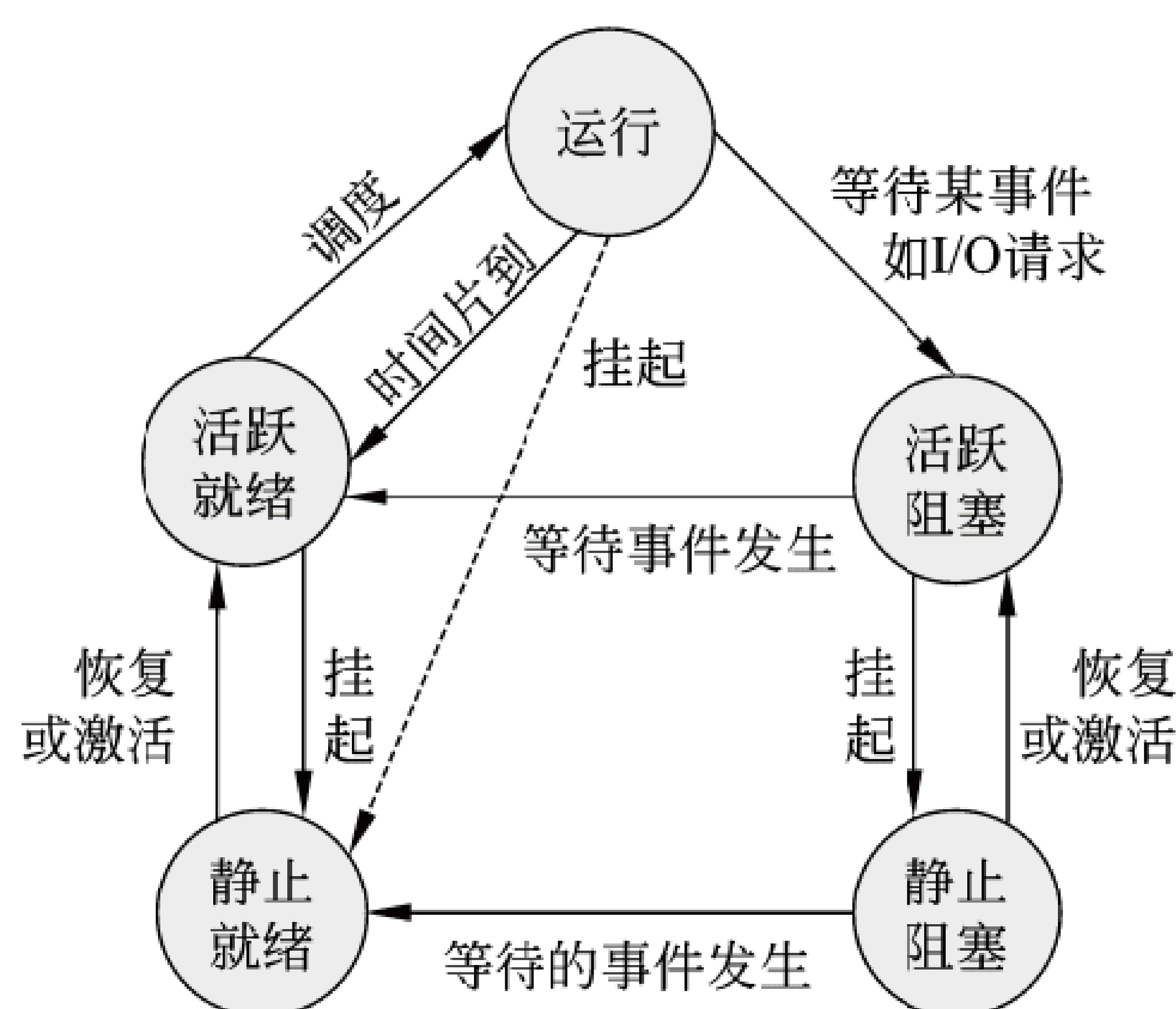


图 2-3　进程状态及其转换

进程控制是指对系统中所有进程从创建到消亡的全过程实施有效的控制。这意味着不仅要控制正在运行的进程，而且还要能创建新的进程，撤销已完成的进程。对进程进行控制的机构是由操作系统内核实现的，大多数操作系统的内核包含支撑功能和资源管理功能。进程控制是通过进程控制原语实现的，进程控制原语主要有创建原语、撤销原语、挂起原语、激活原语、阻塞原语和唤醒原语。

3．进程互斥与同步以及 P，V 操作

1）进程间的同步

异步环境下的一组并发进程之间互发消息、相互合作、互相等待，使得各进程按一定的速度执行的过程称为进程同步，也即同步是使在异步环境下的各进程按一定的顺序和速度执行。

2）进程间的互斥

一组并发进程中的一个或多个程序段，因共享某一公有资源而使得它们必须以一个不允许交叉的顺序执行。也即，互斥要保证临界资源在某一时刻只被一个进程访问。

3）临界资源

系统中有些资源可以供多个进程同时使用，有些资源一次只能供一个进程使用，称为临界资源（Critical Resource，CR），如打印机、公共变量和表格等。

4）临界区管理原则

临界区（Critical Section，CS）是进程中对临界资源实施操作的那段程序。互斥临界区管理的原则是有空即进、无空则等、有限等待、让权等待。

5）信号量机制

1965 年，荷兰学者 Dijkstra 提出的信号量机制是一种卓有成效的进程同步与互斥的

工具。

6）整型信号量与 PV 操作

信号量是一个整型变量，根据控制对象的不同赋不同的值。信号量分为两类。

- 公用信号量：实现进程间的互斥，初值=1 或资源的数目。
- 私用信号量：实现进程间的同步，初值=0 或某个正整数。

信号量 S 的物理意义是：S≥0 表示某资源的可用数，S<0 其绝对值表示阻塞队列中等待该资源的进程数。

PV 操作是实现进程同步与互斥的常用方法。PV 操作是低级通信原语，在执行期间不可分割。其中，P 操作表示申请一个资源，V 操作表示释放一个资源。

P 操作定义：S:=S–1，若 S≥0，则执行 P 操作的进程继续执行；否则若 S<0，则置该进程为阻塞状态（因为无可用资源），并将其插入阻塞队列。

P 操作可用如下过程表示：

```
Procedure P (Var S:Semaphore) ;
          Begin
            S:=S-1;
            If S<0 then W (S)   {执行 P 操作的进程插入等待队列}
          End;
```

V 操作定义：S:=S+1，若 S>0，则执行 V 操作的进程继续执行；否则若 S≤0，则从阻塞状态唤醒一个进程，并将其插入就绪队列，然后执行 V 操作的进程继续执行。

V 操作可用如下过程表示：

```
Procedure V (Var S:Semaphore) ;
          Begin
            S:=S+1;
            If S≤0 then R (S)   {从阻塞队列中唤醒一个进程}
          End;
```

7）利用 PV 操作实现进程的互斥

令信号量 mutex 的初值为“1”，当进入临界区时执行 P 操作，退出临界区时执行 V 操作。则进入临界区的代码段如下：

```
P (mutex)
临界区
V (mutex)
```

8）利用 PV 操作实现进程的同步

进程的同步是由于进程间合作引起的相互制约的问题，要实现进程的同步可用一个信号量与消息联系起来。当信号量的值为 0 时表示希望的消息未产生，当信号量的值为非 0 时表示希望的消息已经存在。假定用信号量 S 表示某条消息，进程可以通过调用 P 操作测试消息是否到达，调用 V 操作通知消息已准备好。

同步问题的经典例子是生产者-消费者问题。相应的程序段形式如下：

- 生产者

Loop
生产一产品 next；
P（Bufempty）；
Next 产品存缓冲区；
V（Buffull）；
endloop

- 消费者

Loop
P（Buffull）；
V（Bufempty）；
从缓冲区中取产品；
使用产品；
endloop

其中，信号量 Bufempty 和 Buffull 分别表示缓冲区中的空单元数和非空单元数，它们的初值分别是 1 和 0。

4．进程通信与管程

1）进程通信

通信（communication）是指进程间的信息交换。根据通信内容可分为控制信息的交换和数据的交换。控制信息的交换称为低级通信，进程的同步与互斥是通过信号量来实现通信的，属于低级信息。数据的交换称为高级通信。高级通信的类型有共享存储系统和消息传递系统和管道通信。高级通信的方式有直接通信和间接通信。

2）管程

汉森（Brinsh Hansen）和霍尔（Hoare）提出了另一种同步机制——管程。

管程是由一些共享数据、一组能为并发进程执行的作用在共享数据上的操作的集合、初始代码以及存取权组成的，也即共享数据及在其上操作的一组过程就构成了管程。进程可以在任何需要资源的时候调用管程，且在任一时刻最多只有一个进程能够真正地进入管程，其他的只能等待。管程提供了一种可以允许多进程安全有效地共享抽象数据类型的机制。

每一个管程都有一个名字，形式如下：

```
Type<管程名>=monitor
          <管程变量说明>;
          define<（能被其他模块引用的）过程名列表>;
          procedure<过程名>（<形式参数表>）
               begin
                    <过程体>;
               end;
          ...
          procedure<过程名>（<形式参数表>）
               begin
                    <过程体>;
               end;
          begin
          <管程的局部数据初始化语句>;
End.
```

5．进程调度与死锁

1）进程调度

进程调度即处理器调度（又称上下文转换），它的主要功能是确定把处理器在什么时候分配给哪一个进程。在某些操作系统中，一个作业从提交到完成需要经历高、中、低三级调度。

2）调度方式与算法

调度方式：调度方式是指当有更高优先级的进程到来时如何分配 CPU。调度方式分为可剥夺和不可剥夺两种。

调度算法：常用的有先来先服务、时间片轮转（round robin）、优先级调度和多级反馈调度算法。

3）死锁

死锁是指两个以上的进程互相都因请求对方已经占有的资源，无限期地等待并无法继续运行下去的现象。

死锁是系统的一种出错状态，它浪费系统资源，还会导致整个系统崩溃，所以应该尽量预防和避免死锁。

4）死锁产生的原因及条件

产生死锁的原因是资源竞争及进程推进顺序非法。产生死锁的 4 个必要条件是互斥条件、请求保持条件、不可剥夺条件和环路条件。

解决死锁的策略：死锁的处理策略主要有 4 种：鸵鸟策略（即不理睬策略）、预防策略（破坏死锁的 4 个必要条件之一）、避免策略（精心地分配资源，动态地回避死锁）、检测与解除死锁（一旦发生死锁，系统不但能检测出，还能解除）。

6．线程

线程是进程中的一个实体，是被系统独立分配和调度的基本单位。在引入了线程的操作系统中，通常一个进程都有若干个线程。线程只拥有一些运行中必不可少的资源，它可与同属一个进程的其他线程共享进程所拥有的全部资源。线程具有许多传统进程所具有的特性，称为轻型进程(Light-Weight Process)；称传统进程为重型进程(Heavy-Weight Process)。线程可创建另一个线程，同一个进程中的多个线程可并发执行。线程也具有就绪、运行、阻塞三种基本状态。

2.1.3　存储管理

存储器是计算机系统的关键性资源，是存放各种信息的主要场所。存储器的发展方向是高速、大容量和小体积。存储管理的主要任务是如何提高主存的利用率、扩充主存以及对主存信息实现有效保护。存储管理的对象是主存储器（简称主存或内存）。

1．存储管理的概念

存储组织的功能是在存储技术和 CPU 寻址技术许可的范围内组织合理的存储结构，使得各层次的存储器都处于均衡的繁忙状态，其依据是访问速度匹配、容量要求和价格等。一般存储器的结构有“寄存器—主存—外存”和“寄存器—缓存—主存—外存”两种，如图 2-4 所示。

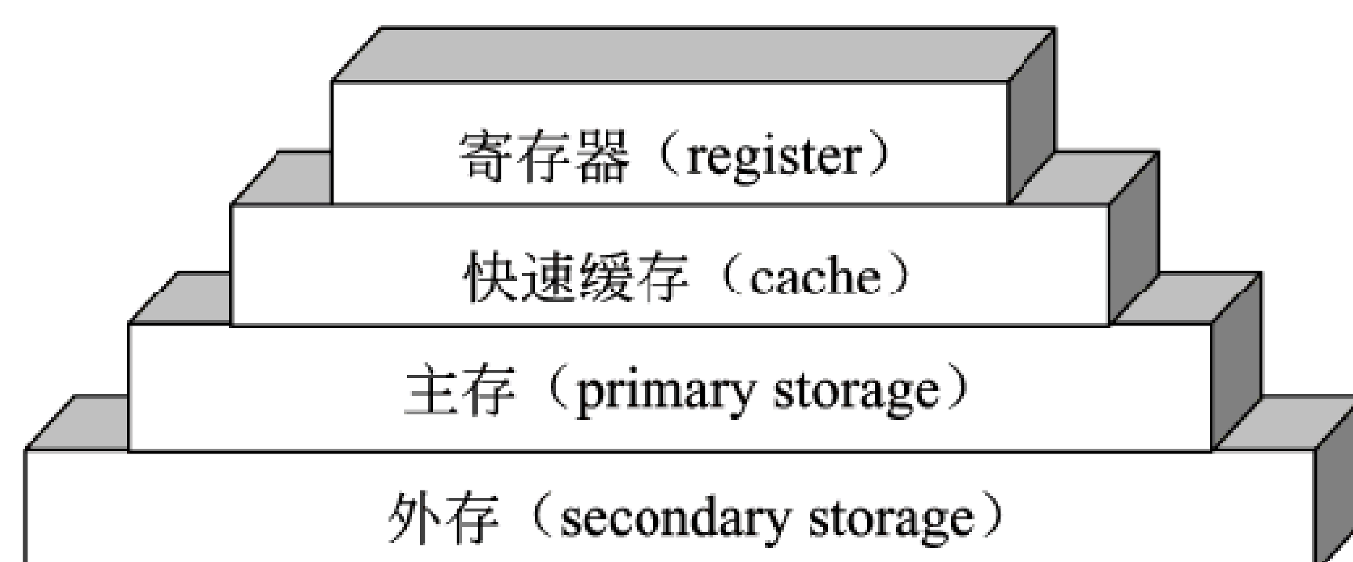


图 2-4　存储器的层次结构

逻辑地址：用户程序经编译后，每个目标模块以 0 为基地址进行的顺序编址，它不是主存中的真实地址，是相对基地址而言的。逻辑地址又称为相对地址、程序地址或虚拟地址。

物理地址：主存中各存储单元的地址，从统一的基地址进行的顺序编址，是主存中的真实地址，可以寻址并实际存在。物理地址又称为绝对地址。

存储空间：简单说来，逻辑地址空间（简称地址空间）是逻辑地址的集合，物理地址空间（简称存储空间）是物理地址的集合。

2．地址重定位

程序的逻辑地址被转换成主存的物理地址的过程称为地址重定位。地址重定位有两种方式：静态重定位（在程序执行之前进行地址重定位，即装入内存时重定位）和动态

重定位（在程序执行期间，在每次存储访问之前进行地址重定位）。

3．存储管理的功能

存储管理的功能有主存储器的分配和回收、提高主存储器的利用率、存储保护、主存扩充。

4．存储管理的方式

存储管理的方式有分区存储管理、分页存储管理、分段存储管理、段页式存储管理和虚拟存储管理。

2.1.4 设备管理

在计算机系统中，输入/输出（I/O）设备、辅存设备及终端设备等都称为外部设备，它们是计算机系统与外界交互的工具，具体负责计算机与外部的输入输出工作。

设备管理的任务是保证在多道程序环境下，当多个进程竞争使用设备时，按一定策略分配和管理各种设备，控制设备的各种操作，完成输入/输出设备与主存之间的数据交换。

设备管理的目标是提高设备的利用率，为用户提供方便统一的界面。

设备管理的主要功能是动态地掌握并记录设备的状态、设备分配和释放、缓冲区管理、实现物理输入/输出设备的操作、提供设备使用的用户接口、设备的访问和控制、输入/输出缓冲和调度。

1．设备的分类

按设备的使用特性分为存储设备、输入/输出设备、终端设备和脱机设备。

从资源分配角度分为独占设备、共享设备和虚拟设备。

按设备的从属关系分为系统设备和用户设备。

按数据组织方式分为块设备（Block Device）和字符设备（Character Device）

按数据传输速率分为低速设备、中速设备、高速设备。

按输入 / 输出对象分为人机通信设备、机机通信设备。

按是否可交互分为非交互设备、交互设备。

2．设备管理的主要技术

- 中断技术
- DMA 技术（Direct Memory Access，DMA）
- 缓冲技术
- 虚设备与 SPOOLING（simultaneous peripheral operations online，外围设备联机）技术

3．设备管理软件

- 中断处理程序
- 设备驱动程序

- 与设备无关的系统软件
- 用户层 I/O 软件

4．数据传输控制方式

设备管理的主要任务之一是控制设备和内存或 CPU 之间的数据传送，常用的数据传送控制方式如下。

- 程序控制方式
- 中断方式
- 直接存储访问方式
- 通道方式

5．磁盘调度算法

磁盘是可供多个进程共享的设备。磁盘调度是使各进程对磁盘的平均访问时间最小。常用的调度算法有先来先服务（first-come first-served，FCFS）、最短寻道时间优先（Shortest Seek Time First，SSTF）、扫描算法（SCAN）。

2.1.5　文件管理

文件（File）是具有符号名的、在逻辑上具有完整意义的一组相关信息项的集合。文件名的格式和长度因系统而异，操作系统根据文件名对其进行控制和管理。

文件管理系统是操作系统中对文件进行统一管理的一组软件和相关数据（即被管理的文件）的集合，简称文件系统。

文件系统的功能按名存取、统一用户接口、并发访问和控制、安全性控制、优化性能以及差错恢复。

1．文件的类型

按文件性质和用途可分为系统文件、库文件和用户文件。

按文件的安全属性可分为只读文件、读写文件、可执行文件和不保护文件。

按文件的组织形式可分为普通文件、目录文件、设备文件（特殊文件）。

按信息保存期限可分为临时文件、档案文件和永久文件。

按信息流向可分为输入文件、输出文件、输入/输出文件。

2．文件的结构和组织

文件的结构是指文件的组织形式。从用户角度看到的文件组织形式称为文件的逻辑结构，从实现角度看到的文件在存储设备上的存放方式，称为文件的物理结构。

文件的逻辑结构有结构的记录文件和无结构的字符流文件。

文件的物理结构有连续结构、链接结构、索引结构、多个物理块的索引表。

3．文件访问方法

文件的访问方法是指读写文件存储设备上的一个物理块的方法。常用的访问方法有顺序访问和随机访问。顺序访问是指对文件中的信息按顺序依次读写的方式；随机访问

是指对文件中的信息可以按任意的次序随机地读写文件中的信息。

4．文件存储设备管理

文件是存储在文件存储设备上的，文件存储设备具有大容量、被多用户共享、多次被占用和释放的特点，因此，文件系统必须对文件存储设备上的空闲空间进行组织和管理，包括对空闲空间的组织、分配与回收等。常用的空闲空间管理方法有位图法、索引法和链接法。

5．文件控制块和文件目录

文件控制块是系统为每个文件设置的用于描述和控制文件的数据结构，它是文件存在的唯一标志，简称为（File Control Block，FCB）。FCB 一般包含基本信息、位置信息、存取控制信息和使用信息。

文件目录是文件控制块的有序集合。常见的文件目录结构有一级目录结构、二级目录结构和多级目录结构。

6．文件的使用

工作目录也称当前目录。每个用户都有自己的工作目录，任一目录节点都可以被设置为工作目录，文件系统允许用户随时改变自己的工作目录。

文件系统提供了一组专门用于目录和文件管理的命令。如目录管理命令：建立目录、显示工作目录、改变目录、删除目录；文件控制命令：建立文件、删除文件、打开文件、关闭文件、改文件名、改变文件属性；文件存取命令：读写文件、显示文件内容、复制文件等。

文件的共享是指不同的用户使用同一文件，它是不同用户完成同一任务的必须的功能。

文件的安全是指文件的保密和保护，即限制非法用户使用和破坏文件。文件的安全管理措施常常在系统级、用户级、目录级和文件级上实施。

2.1.6 作业管理

1．作业管理和作业控制

作业是系统为完成一个用户的计算任务（或一次事务处理）所做的工作总和。它由程序、数据和作业说明书三部分组成。作业管理程序是操作系统中用来控制作业进入、执行和撤销的一组程序。

用户作业可以采用脱机和联机两种控制方式控制作业运行。作业控制块 JCB 是记录与该作业有关的各种信息的登记表。JCB 是作业存在的唯一标志，包括用户名、作业名、状态标志等信息。在输入井中，通常将作业控制块排成一个或多个队列，称为作业后备队列，也就是说作业后备队列是由若干个 JCB 组成的。

2．作业状态及转换

作业的 4 种状态为提交、后备、执行和完成。作业的状态及其转换如图 2-5 所示。

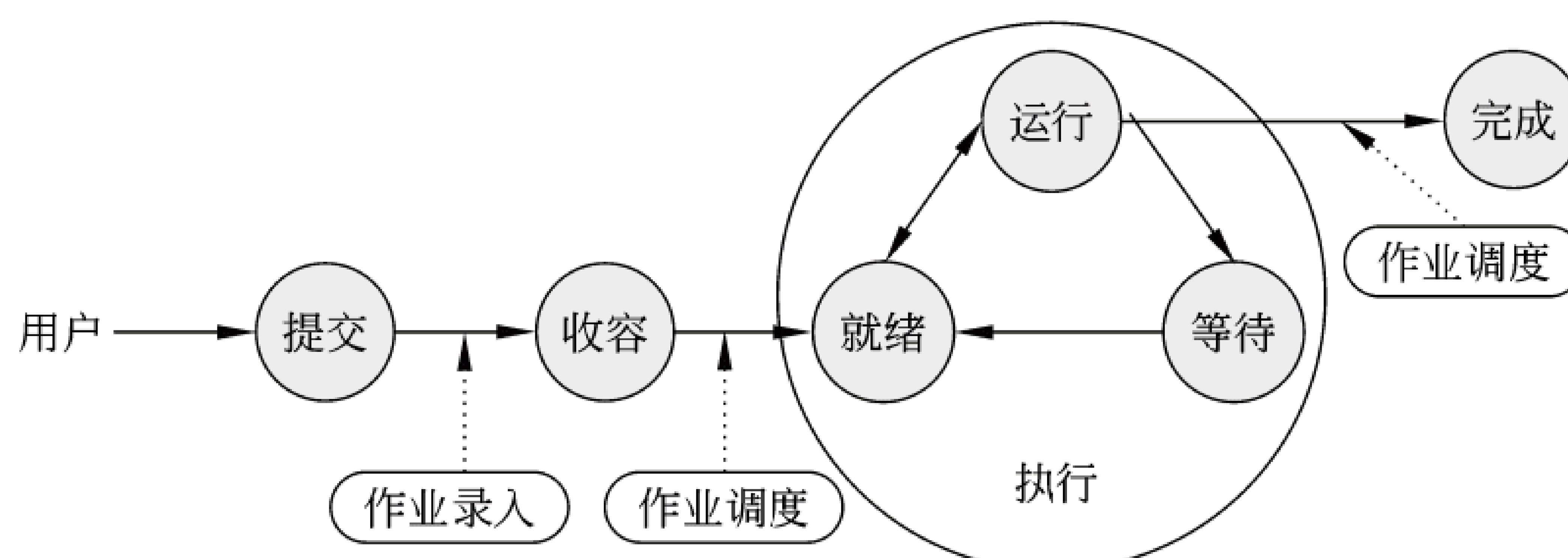


图 2-5　作业的状态及其转换

3．作业调度及其常用调度算法

作业调度是完成从后备状态到执行状态的转变及从执行状态到完成状态的转变。常用的作业调度算法有先来先服务、短作业优先、响应比高优先、优先级调度算法和均衡调度算法。在一个以批量处理为主的系统中，通常用平均周转时间或平均带权周转时间来衡量作业调度算法的性能。

4．用户界面

用户界面是计算机中实现用户与计算机通信的软、硬件的总称。用户界面也称用户接口或人机界面。

用户界面的硬件部分包括用户向计算机输入数据或命令的输入装置及由计算机输出供用户观察或处理的输出装置。目前常用的输入 / 输出装置有键盘、鼠标、显示器和打印机等。用户界面的软件部分包括用户与计算机相互通信的协议、约定、操纵命令及其处理软件。常用的人机通信方法有命令语言、菜单选项、图符驱动、表格填充、视窗操作及直接操纵等。

2.1.7　网络操作系统

1．网络操作系统

网络操作系统（Network Operating System，NOS）是使网络中各计算机能方便而有效地共享网络资源，为网络用户提供所需的各种服务的软件和有关规则的集合。通常的操作系统具有处理机管理、存储器管理、设备管理及文件管理，而网络操作系统除了具有上述的功能外，还具有提供高效、可靠的网络通信能力和提供多种网络服务的功能。

2．网络操作系统的特征

网络操作系统的特征包括硬件独立性、多用户支持、支持网络实用程序及其管理功能、多种客户端支持、提供目录服务以及支持多种增值服务等。

3．网络操作系统的分类

网络操作系统分为集中模式、客户机/服务器模式和对等模式（peer-to-peer）。

现代操作系统已把网络功能包含到操作系统的内核中，作为操作系统核心功能的一个组成部分，并由此决定了不同网络的应用领域及方向。目前最流行的网络操作系统主要有 NetWare 系列、Windows 系列、UNIX 和 Linux 及相应的服务软件。

2.1.8 常见操作系统简介

1．Unix 系统

UNIX 操作系统是由美国贝尔实验室发明的一种多用户、多任务的分时操作系统。现已广泛运行于中型机和小型机等各种环境，用于大型信息系统的关键业务处理，如数据库和 Internet 主机。UNIX 结构如图 2-6 所示。UNIX 最内层硬件提供基本服务，内核提供全部应用程序所需的各种服务。

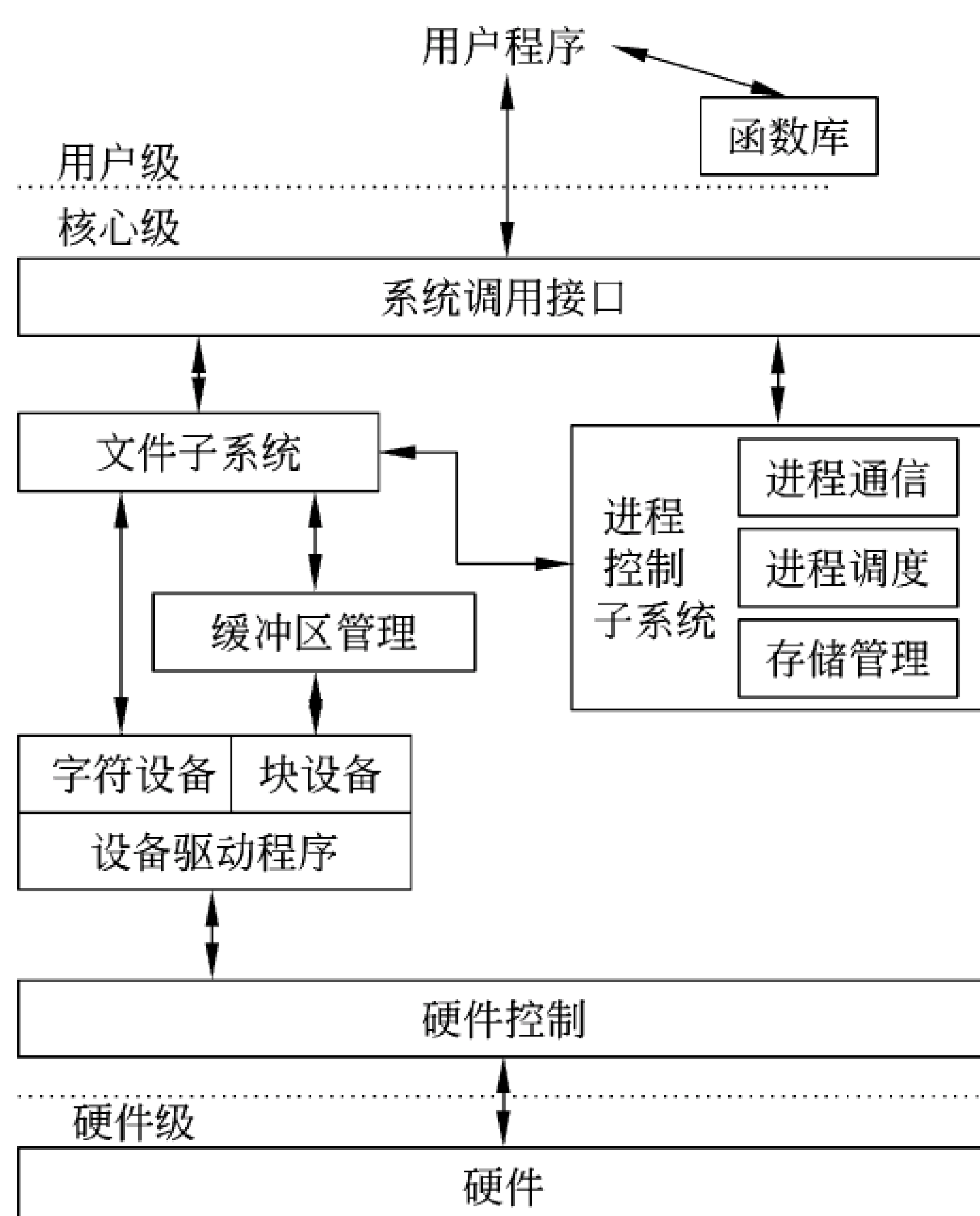


图 2-6 UNIX 系统结构

目前，UNIX 通常与服务硬件产品集成在一起，具有代表性的有 IBM 公司的 AIXUNIX、SUN 公司的 SolarisUNIX 和 HP 公司的 HPUNIX 等。

2．Windows 家族

Microsoft 公司的 Windows 操作系统家族产品有 Windows 95、Windows 98、Windows ME、Windows 2000 Professional、Windows XP、Windows Server 2003 和 Windows

Vista 等。

Windows Server 2003 是继 Windows XP 后 Microsoft 发布的又一个产品，它在 Windows 2000 Server 的基础上增加了许多新功能，包括配置流程向导、远程桌面连接、Internet 信息服务（IIS6.0）、简单的邮件服务器（POP3）、流式媒体服务器（Windows Media Services，WMS）、系统关闭事件跟踪等功能。

3．Linux 系统

1991 年，芬兰赫尔辛基大学的学生 Linus Torvalds 利用互联网，发布了他在 i386 个人计算机上开发的 Linux 操作系统内核的源代码，创建了具有全部 UNIX 特征的 Linux 操作系统。Linux 是一个支持多用户、多任务、多进程、实时性较好的、功能强大而稳定的操作系统，也是目前运行硬件平台最广泛的操作系统。Linux 是以互联网和开放源码为基础的，许多系统软件的设计专家们都利用互联网对它进行了改进和提高。近年来，Linux 得到了包括 IBM、COMPAQ、HP、Oracle、Sybase、Informix 在内的许多著名软硬件公司的支持，目前 Linux 已全面进入应用领域。

RedHatLinux 是目前世界上使用最多的 Linux 操作系统家庭成员，它提供了丰富的软件包，具有强大的网络服务、多媒体应用和安全管理等功能。

2.2　数据库系统基础知识

数据库（DataBase，DB）是指长期储存在计算机内的、有组织的、可共享的数据集合。数据库系统（DataBase System，DBS）从广义上讲是由数据库、硬件、软件和人员组成，管理的对象是数据。数据库管理系统（DataBase Management System，DBMS）是一种操纵和管理数据库的大型软件，用于建立、使用和维护数据库。主要功能有数据定义、数据库操作、数据库运行管理、数据组织、存储和管理、数据库的建立与维护及其他功能。DBMS 通常分为三类：关系数据库系统（Relation DataBase Systems，RDBS)、面向对象的数据库系统（Object-Oriented DataBase system，OODBS)、对象关系数据库系统（Object-Oriented Relation DataBase system，ORDBS）。

数据库系统采用三级模式结构，如图 2-7 所示。数据库系统在三级模式间提供了两级映像：模式 / 内模式映像、外模式 / 模式映像来保证数据库中数据具有较高的逻辑独立性和物理独立性。

2.2.1　关系数据库基础

1．数据库的结构与模式

数据库结构的基础是数据模型，是用来描述数据的一组概念和定义。数据模型的三要素是数据结构、数据操作以及数据的约束条件。常用的数据模型有概念数据模型、基本数据模型以及面向对象模型。

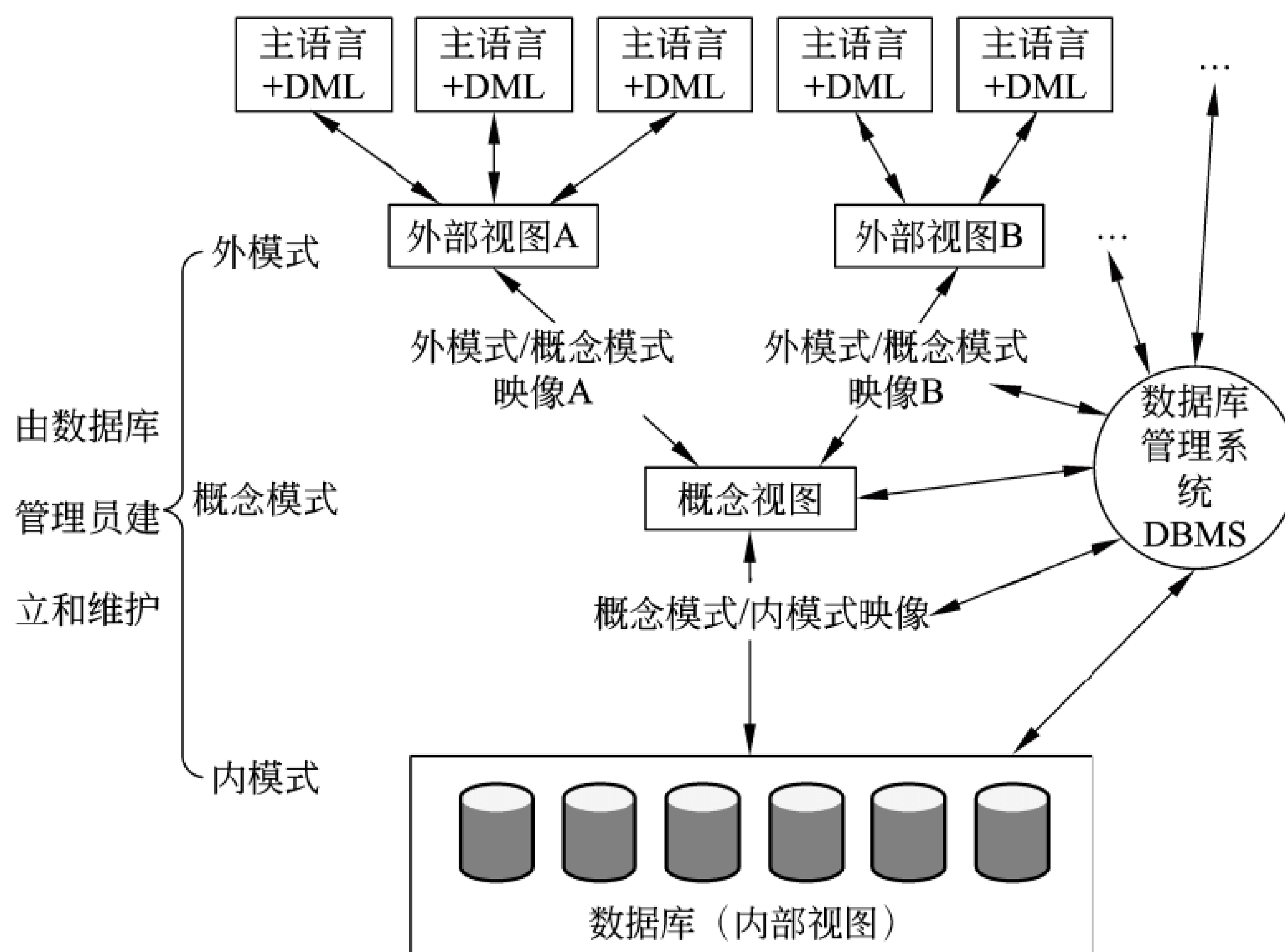


图 2-7 数据库系统体系结构

关系数据模型由关系数据结构、关系操作集合和关系完整性约束三大要素组成。

2．实体-联系（E-R）模型

概念模型中最常用的方法为实体-联系方法，简称 E-R 方法，主要概念有实体、联系和属性。该方法直接从现实世界中抽象出实体和实体间的联系（如图 2-8 所示）。

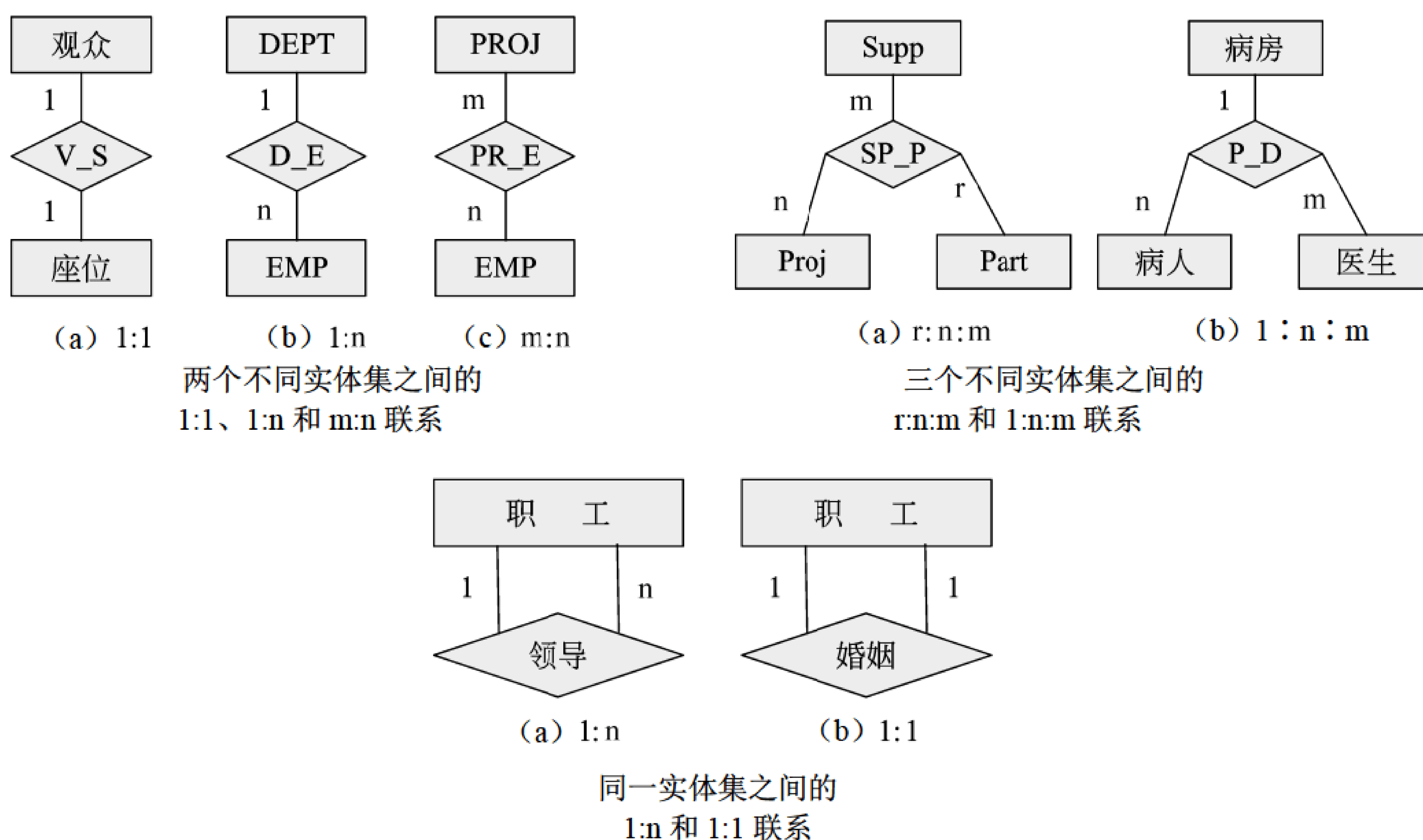


图 2-8 实体和实体间的联系

E-R 图的主要构件如表 2-1 所示。

表 2-1 E-R 图的主要构件

构 件		说 明
矩形		表示实体集
菱形		表示联系集
椭圆		表示属性
线段		将属性与相关的实体集连接，或将实体集与联系集相连
双椭圆		表示多值属性
虚椭圆		表示派生属性
双线		表示一个实体全部参与到联系集中

某学校教学管理的 E-R 模型如图 2-9 所示。

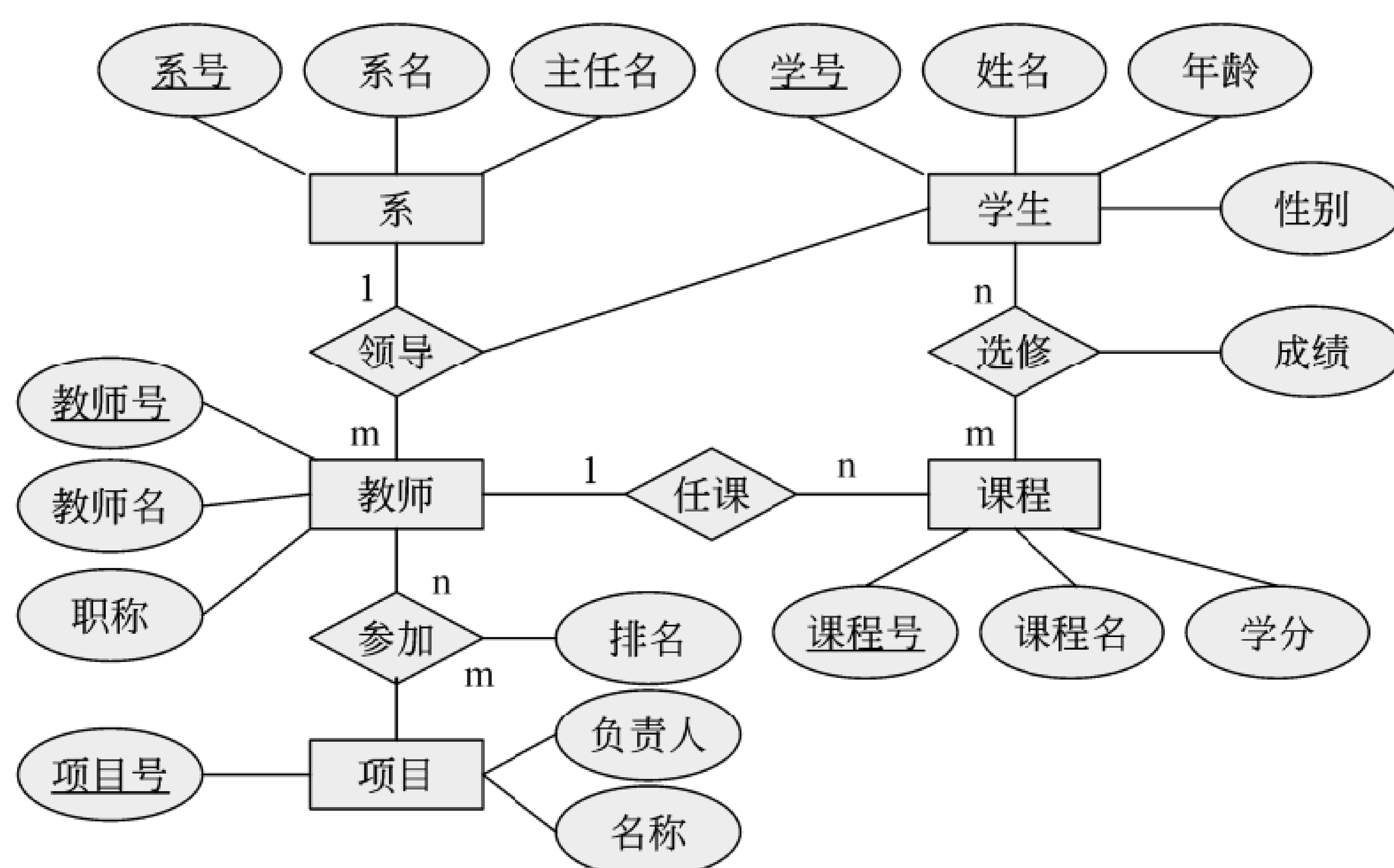


图 2-9 某学校教学管理 E-R 模型

特别需要指出的是，E-R 模型强调的是语义，与现实世界的问题密切相关。扩充的 E-R 模型包括弱实体、特殊化、概括和聚集等。

3．数据的规范化

规范化理论研究的是关系模式中各属性之间的依赖关系及其对关系模式性能的影响。关系数据库设计理论的核心是数据间的函数依赖，衡量的标准是关系规范化的程度及分解的无损连接和保持函数依赖性。

数据依赖是通过一个关系中属性间值的相等与否体现出来的数据间的相互关系，是现实世界属性间联系和约束的抽象，是数据内在的性质，是语义的体现。函数依赖则是一种最重要、最基本的数据依赖。包括函数依赖、非平凡的函数依赖、平凡的函数依赖、

完全函数依赖、部分函数依赖、传递依赖、码、主属性和非主属性、外码、值依赖定义、函数依赖的公理系统（Armstrong 公理系统）。

范式是关系模型满足的确定约束条件。范式有 1NF（第一范式）、2NF（第二范式）、3NF（第三范式）、BCNF（巴克斯范式）、4NF（第四范式）和 5NF，其中 1NF 级别最低。这几种范式之间有 $5NF \subset 4NF \subset BCNF \subset 3NF \subset 2NF \subset 1NF$ 成立。

关系模型的规范化是指把一个低一级范式的关系模式转换成若干个高一级范式的关系模式的过程。关系数据库设计的方法之一就是设计满足适当范式的模式，并通过判断分解后的模式达到第几范式来评价模式规范化的程度。

4．事务管理

事务是一个操作序列，这些操作"要么都做，要么都不做"，事务是数据库环境中不可分割的逻辑工作单位。事务和程序是两个不同的概念，一般一个程序可包含多个事务。

事务的 4 个特性：原子性（atomicity）、一致性（consistency）、隔离性（isolation）和持久性（durability）。这 4 个特性也称为事务的 ACID 性质。

在 SQL 语言中事务定义的语句有三条：BEGIN TRANSACTION 事务开始、COMMIT 事务提交和 ROLLBAK 事务回滚。

5．并发控制

并发操作是指在多用户共享的系统中，用户可能同时对同一数据进行操作。并发操作带来的问题是数据的不一致，主要有丢失更新、不可重复读和读脏数据。其主要原因是事务的并发操作破坏了事务的隔离性。DBMS 的并发控制子系统负责协调并发事务的执行，保证数据库的完整性不受破坏，避免用户得到不正确的数据。

并发控制的主要技术是封锁。封锁的类型有排他锁（简称 X 锁或写锁）和共享锁（简称 S 锁或读锁）。并发控制还与三级封锁协议、活锁与死锁、并发调度的可串行性、两段封锁协议、封锁的粒度、事务的嵌套等有关。

6．数据库的备份与恢复

在数据库的运行过程中，难免会出现计算机系统的软、硬件故障，从而影响数据库中数据的正确性，甚至破坏数据库，使数据库中全部或部分数据丢失。因此，保护数据库的关键技术在于建立冗余数据，即备份数据。建立冗余数据的方法是进行数据转储和建立日志文件。数据的转储分为静态转储和动态转储、海量转储和增量转储。如何在系统出现故障后能够及时使数据库恢复到故障前的正确状态，就是数据库恢复技术。

数据库的 4 类故障是事务故障、系统故障、介质故障及计算机病毒。

事务故障的恢复一般有两个操作：撤销事务（UNDO）和重做事务（REDO）。

介质故障的恢复需要数据库管理员（DataBase Administrator，DBA）的参与，装入数据库的副本和日记文件副本，再由系统执行撤销和重做操作。

在一个数据库系统中，这两种方法一般是同时采用的。为了避免磁盘介质出现故障影响数据库的可用性，许多 DBMS 提供数据库镜像功能用于数据库恢复，数据库镜像是

通过复制数据实现的，但频繁地复制数据会降低系统的运行效率，因此实际应用中往往只对关键的数据和日志文件镜像。

2.2.2　关系数据库设计

数据库设计是指对于一个给定的应用环境，构造最优的数据库，建立数据库及其应用系统，使之能有效地存储数据，满足各种用户的需求。数据库设计包括结构特性的设计和行为特性的设计两方面的内容。

1．数据库设计的特点

数据库设计的很多阶段都可以和软件工程的各阶段对应起来，数据库设计的特点有：从数据结构即数据模型开始，并以数据模型为核心展开，这是数据库设计的一个主要特点；静态结构设计与动态行为设计分离；试探性；反复性和多步性。

2．数据库设计的方法

目前已有的数据库设计方法可分为 4 类，即直观设计法、规范设计法、计算机辅助设计法和自动化设计法。常用的有基于 3NF 的设计方法、方法、基于实体联系（E-R）模型的数据库设计方法、基于视图概念的数据库设计方法、面向对象的关系数据库设计方法、计算机辅助数据库设计方法、敏捷数据库设计方法等。

3．数据库设计的基本步骤

数据库设计分为需求分析、概念结构设计、逻辑结构设计、物理结构设计、应用程序设计和运行维护 6 个阶段，如图 2-10 所示。

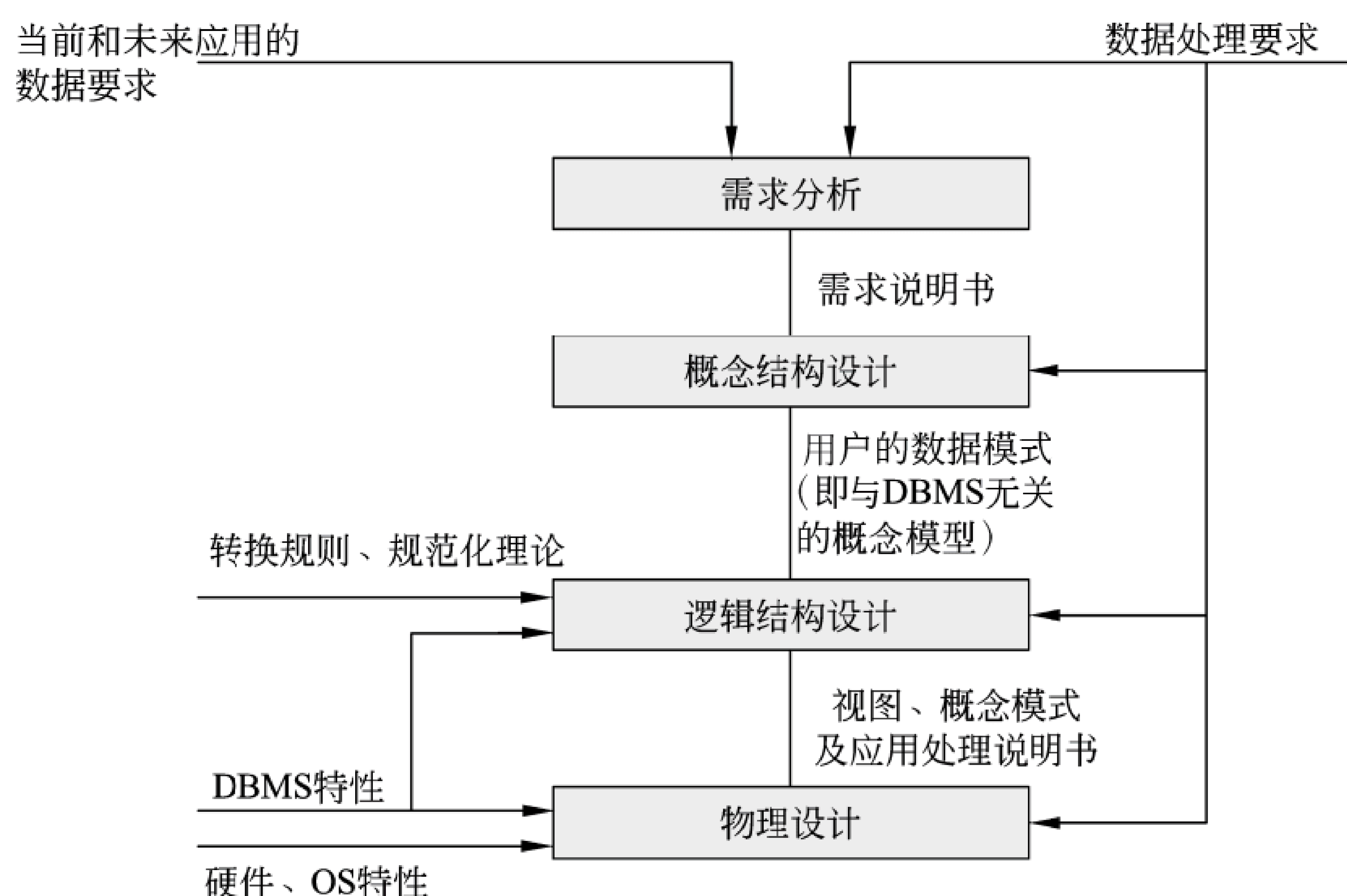


图 2-10　数据库的设计步骤

1）需求分析

需求分析阶段的任务是：对现实世界要处理的对象（组织、部门、企业等）进行详细调查，在了解现行系统的概况、确定新系统功能的过程中，收集支持系统目标的基础数据及其处理方法。需求分析是在用户调查的基础上，通过分析逐步明确用户对系统的需求，包括数据需求和围绕这些数据的业务处理需求。

在需求分析中，通过自顶向下、逐步分解的方法分析系统。分析的结果用数据流图（Data Flow Diagram，DFD）进行图形化的描述，并用一些规范的表格对数据分析结果和描述做补充，最后形成需求说明书。

2）概念结构设计

数据库概念结构设计是在需求分析的基础上，依照需求分析中的信息需求，对用户信息加以分类、聚集和概括，建立信息模型，并依照选定的数据库管理系统软件，把它们转换为数据的逻辑结构，再依照软硬件环境，最终实现数据的合理存储。这一过程也称为数据建模。

数据库概念结构设计的策略通常有自顶向下、自底向上、逐步扩张、混合策略，最常用的是自底向上策略。

设计数据库概念模型的最著名、最常用的方法是 P.P.S.chen 于 1976 年提出的“实体-联系方法”（Entity Relationship Approach），简称 E-R 方法。采用 E-R 方法的数据库概念结构设计可分为三步：设计局部 E-R 模型、设计全局 E-R 模型以及全局 E-R 模型的优化。

3）逻辑结构设计

逻辑结构设计是在概念结构设计基础上进行的数据模型设计，可以是层次、网状模型和关系模型。逻辑结构设计阶段的主要任务是确定数据模型、将 E-R 图转换为指定的数据模型、确定完整性约束、确定用户视图。

4）物理结构设计

数据库在物理设备上的存储结构与存取方法称为数据库的物理结构。数据库的物理结构设计是对已确定的数据库逻辑结构，利用 DBMS 所提供的方法、技术，以较优的存储结构和数据存取路径、合理的数据存放位置以及存储分配，设计出一个高效的、可实现的数据库物理结构。

一般来说，物理结构设计要做的工作有存储记录的格式设计、存储结构设计、存取方法设计和确定系统配置。

5）数据库应用程序设计

数据库应用系统开发是 DBMS 的二次开发，一方面是对用户信息的存储，另一方面就是对用户处理要求的实现。

数据库应用程序设计要做的工作有选择设计方法、制定开发计划、选择系统架构、设计安全性策略。在应用程序设计阶段，设计方法有结构化设计方法和面向对象设计方法两种；安全性策略主要是指硬件平台、操作系统、数据库系统、网络及应用系统的安全。

数据库应用系统的实现是根据设计、由开发人员编写代码程序来完成的，包括数据库的操作程序和应用程序。作为关系数据库标准语言，SQL 已经被大量的 DBMS 系统所使用。

6）数据库运行和维护

数据库的正常运行和优化也是数据库设计的内容之一。在数据库运行维护阶段要做的工作主要有数据库的转储和恢复，数据库的安全性和完整性控制，数据库性能的监督、分析和改造，数据库的重组和重构等。

2.2.3　分布式数据库系统

1．分布式数据库的概念

分布式数据库系统（Distributed Database System，DDBS）是针对地理上分散，而管理上又需要不同程度集中管理的需求而提出的一种数据管理信息系统。满足分布性、逻辑相关性、场地透明性和场地自治性的数据库系统被称为完全分布式数据库系统。

分布式数据库系统的特点是数据的集中控制性、数据独立性、数据冗余可控性、场地自治性和存取的有效性。

2．分布式数据库的体系结构

我国在多年研究与开发分布式数据库及制定《分布式数据库系统标准》中，提出了把分布式数据库抽象为 4 层的结构模式，如图 2-11 所示。这种结构模式得到了国内外一定程度的支持和认同。

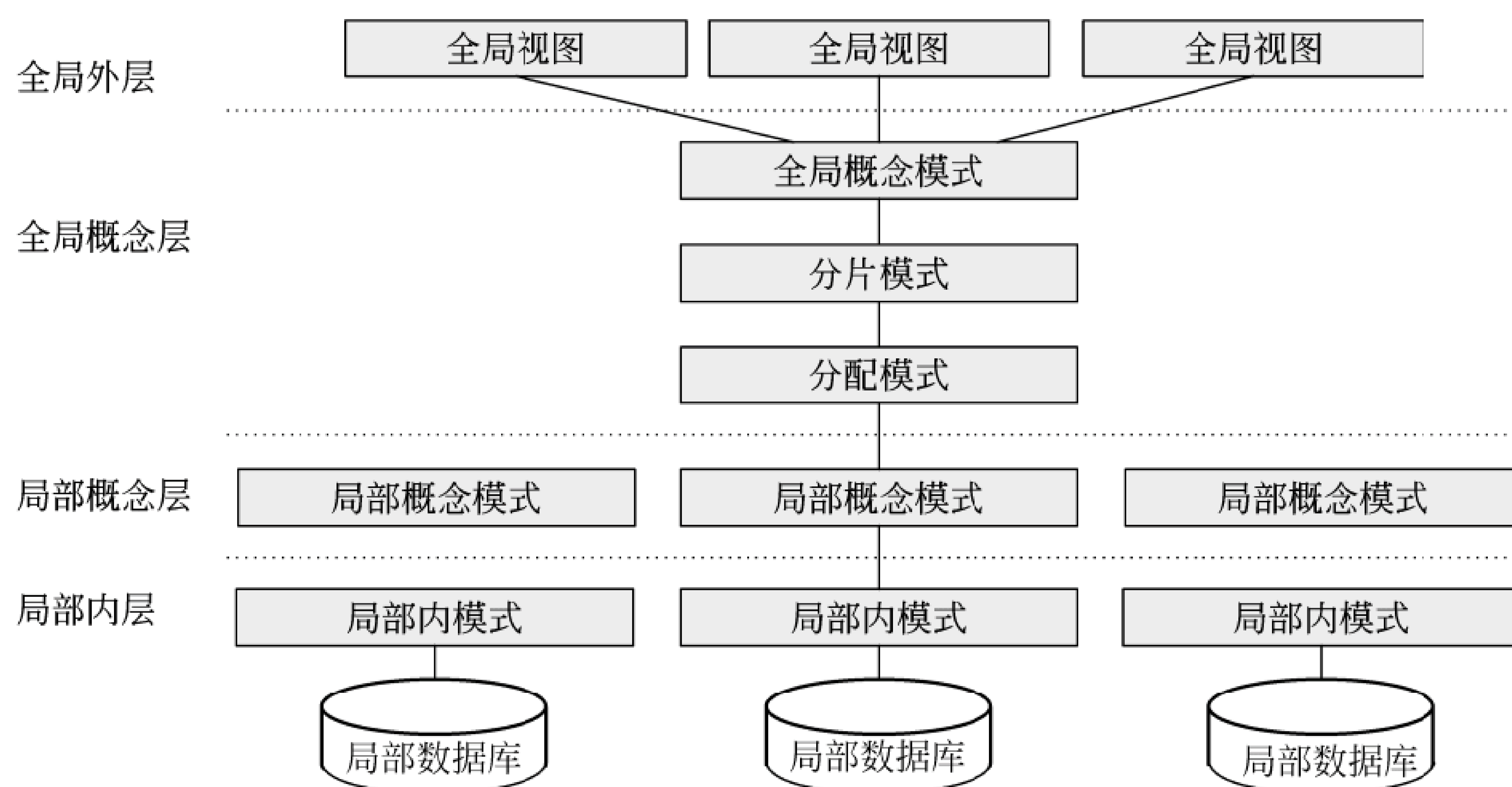


图 2-11　分布式数据库结构模式图

4 层模式划分为全局外层、全局概念层、局部概念层和局部内层，在各层间还有相应的层间映射。这种 4 层模式适用于同构型分布式数据库系统，也适用于异构型分布式

数据库系统。

3．分布式数据库系统的应用

分布式数据库的应用领域有分布式计算、Internet 应用、数据仓库、数据复制以及全球联网查询等，Sybase 公司的 Replication Server 即是一种典型的分布式数据库系统。

2.2.4 商业智能

1．商业智能基本概念

商业智能（Business Intelligence，BI）是企业对商业数据的搜集、管理和分析的系统过程，目的是使企业的各级决策者获得知识或洞察力，帮助他们做出对企业更有利的决策。它是数据仓库、联机分析处理（Online Analytical Processing，OLAP）和数据挖掘等相关技术走向商业应用后形成的一种应用技术。

商业智能系统主要实现将原始业务数据转换为企业决策信息的过程。它主要包括数据预处理、建立数据仓库、数据分析及数据展现 4 个主要阶段。

一般认为数据仓库、联机分析处理和数据挖掘技术是商业智能的三大组成部分。

2．数据仓库

1）数据仓库的概念与特性

著名的数据仓库专家 W.H.Inmon 在 Building the Data Warehouse 一书中将数据仓库定义为：数据仓库（Data Warehouse）是一个面向主题的（Subject Oriented）、集成的（Integrate）、相对稳定的（Non-Volatile），且随时间变化的（Time Variant）数据集合，支持管理部门的决策过程。

数据仓库的关键特征为面向主题、集成的、非易失的、时变的。

2）数据仓库的结构

数据仓库采用三层结构，底层是数据仓库服务器、中间层是 OLAP 服务器、顶层是前端工具，如图 2-12 所示。

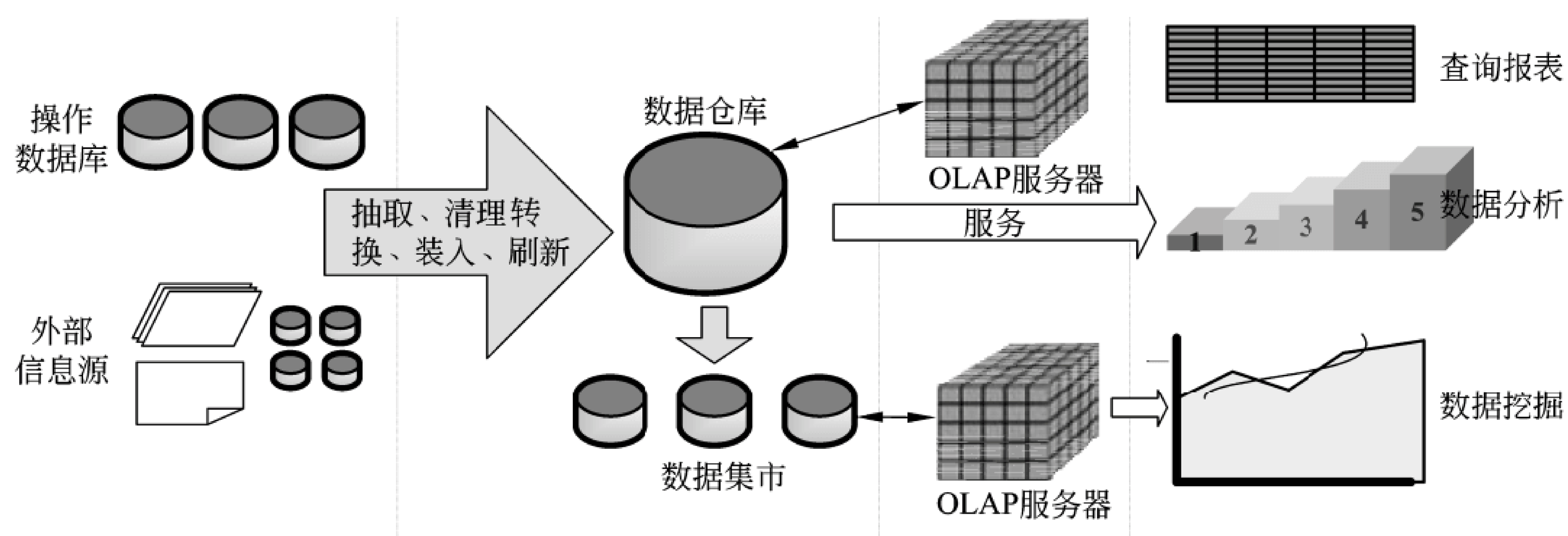


图 2-12　数据仓库体系结构

3）数据仓库的实现方法

数据仓库的实现步骤有规划、需求研究、问题分析、数据的抽取清洗集成装载、数据仓库设计、数据仓库管理、分析报表查询、数据仓库性能优化及数据仓库的部署发布等几个步骤。实现方法有自顶向下方法、自底向上方法及二者混合方法。

对于开发数据仓库系统，一个推荐的方法是以递增、进化的方式实现数据仓库，如图 2-13 所示。

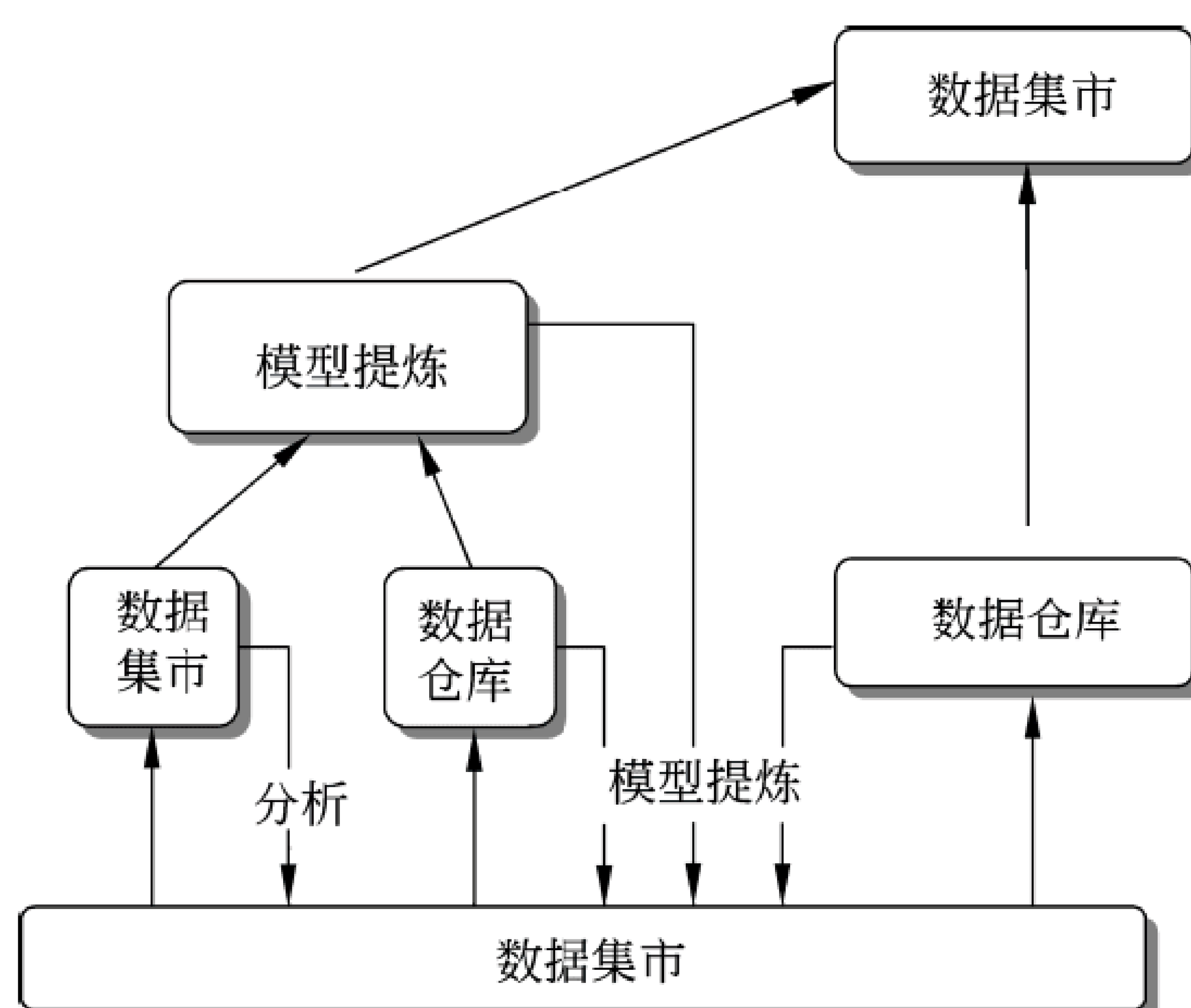


图 2-13　数据仓库开发的推荐方法

3．多维分析海量数据分析器——OLAP

对于 TB 级的海量数据，联机分析处理 OLAP 利用多维的概念，提供了切片、切块、下钻、上卷和旋转等多维度分析与跨维度分析功能。

OLAP 系统架构主要分为基于关系数据库的 ROLAP（Relational OLAP）、基于多维数据库的 MOLAP（Multidimensional OLAP）和基于混合数据组织的 HOLAP（Hybrid OLAP）三种。

4．数据挖掘

从技术上来看，数据挖掘（data mining）是从大量的、不完全的、有噪声的、模糊的、随机的数据中，提取隐含在其中的、人们事先不知道的、但又是潜在有用的信息和知识的过程。从商业的角度来看，数据挖掘是一种新的商业信息处理技术，其主要特点是对大量业务数据进行抽取、转换、模型化处理，从中提取辅助商业决策的关键性数据。我们采用数据挖掘的广义观点：数据挖掘是从存放在数据库、数据仓库或其他信息库中的大量数据中挖掘有趣知识的过程。

数据挖掘与传统的数据分析（如查询、报表、联机应用分析）的本质区别是：数据挖掘是在没有明确假设的前提下去挖掘信息、发现知识。数据挖掘所得到的信息应具有先知、有效和实用三个特征。

1）数据挖掘的功能

数据挖掘的目标是从数据库中发现隐含的、有意义的知识，主要功能有 5 类：自动预测趋势和行为、关联分析、聚类、概念描述和偏差检测。

2）常用的数据挖掘技术

常用的数据挖掘技术包括关联分析、序列分析、分类分析、聚类分析、预测以及时间序列分析等。

3）数据挖掘的流程

数据挖掘的流程为确定挖掘对象、准备数据、建立模型、数据挖掘、结果分析和知识应用。

4）数据挖掘的应用

从目前情况来看，数据挖掘的热点包括空间数据库的挖掘、多媒体数据库的挖掘、时序数据和序列数据的挖掘、文本数据库的挖掘、Web 挖掘。

数据挖掘的应用领域有生物医学和 DNA 的数据挖掘、金融业中的数据挖掘、零售业中的数据挖掘、电信业中的数据挖掘、视频和音频数据挖掘、科学和统计数据挖掘等。

商用数据挖掘的事例有 Intelligent Miner、Enterprise Miner、MineSet、Clementine、DBMiner 等。

2.2.5 常见的数据库管理系统

1．Oracle

Oracle 是一种适用于大型、中型和微型计算机的关系数据库管理系统。Oracle 的结构包括数据库的内部结构、外存储结构、内存储结构和进程结构。在 Oracle 中，数据库不仅指物理上的数据，还包括处理这些数据的程序，即 DBMS 本身。Oracle 使用 PL/SQL（Procedural Language/SQL）语言执行各种操作。Oracle 除了以关系格式存储数据外，Oracle 8 以上的版本还支持面向对象的结构（如抽象数据类型）。

Oracle 产品主要包括数据库服务器、开发工具和连接产品三类。Oracle 还提供了一系列的工具产品，如逻辑备份工具 Export、Import 等。

2．IBM DB2

DB2 是 IBM 的一种分布式数据库解决方案。简单地说，DB2 就是 IBM 开发的一种大型关系型数据库平台，它支持多用户或应用程序在同一条 SQL 语句中查询不同 Database 甚至不同 DBMS 中的数据。

DB2 核心数据库的特色有支持面向对象的编程、支持多媒体应用程序、备份和恢复功能、支持存储过程和触发器、支持 SQL 查询、支持异构分布式数据库访问、支持数据复制。

DB2 采用多进程多线索体系结构，可运行于多种操作系统之上。IBM 还提供了 Visualizer、Visualage、Visualgen 等开发工具。

3．Sybase

Sybase 是美国 SYBASE 公司在 20 世纪 80 年代中推出的客户机/服务器（Client/Server，CLS）结构的关系数据库系统，也是世界上第一个真正的基于客户机/服务器结构的 RDBMS 产品。

Sybase 数据库主要由三部分组成：进行数据库管理和维护的联机的关系数据库管理系统 Sybase SQLServer，支持数据库应用系统建立与开发的一组前端工具 Sybase SQLToolset，可把异构环境下其他厂商的应用软件和任何类型的数据连接在一起的接口 Sybase OpenClient/OpenServer。

Sybase 提供了 Sybase Adaptive Server Enterprise 高性能企业智能型关系数据库管理系统、EAServer 电子商务解决方案应用服务器、系统分析设计工具 PowerDesigner 和应用开发工具 PowerBuilder。

4．Microsoft SQL Server

Microsoft SQL Server 是一种典型的关系型数据库管理系统，可运行于多个操作系统上，它使用 Transact-SQL 语言完成数据操作。

SQL Server 的基本服务器组件包括 Open Data Services、MS SQL Server、SQL Server Agent 和 MSDTC（Microsoft Distributed Transaction Coordinator）。

SQL Server 数据平台包括以下工具：关系型数据库、复制服务、通知服务、集成服务、分析服务、报表服务、管理工具和开发工具。

2.3　计算机网络基础知识

2.3.1　网络概述

计算机网络是指利用通信设备和线路将地理位置分散的、功能独立的计算机系统或由计算机控制的外部设备连接起来，在网络操作系统的控制下，按照约定的通信协议进行信息交换、实现资源共享的系统。计算机网络的组成元素有网络结点和通信链路。计算机网络的功能有数据通信、资源共享、负载均衡和高可靠性。

计算机网络按通信距离可分为广域网（WAN）、局域网（LAN）和城域网（MAN）；按信息交换方式可分为电路交换网、分组交换网和综合交换网；按网络拓扑结构可分为星形网、树形网、环形网和总线网；按通信介质可分为双绞线网、同轴电缆网、光纤网和卫星网等；按传输带宽可分为基带网和宽带网；按使用范围可分为公用网和专用网；按速率可分为高速网、中速网和低速网；按通信传播方式可分为广播式和点到点式；按使用方式可分为校园网和企业网；按连接范围可分为内联网和外联网；按网络提供的服务可分为通信网和信息网。

1．开放系统互连参考模型

开放系统是指遵从国际标准的、能够通过互连而相互作用的系统。系统之间的相互作用只涉及系统的外部行为，而与系统内部的结构和功能无关。国际标准化组织（International Standard Organization，ISO）公布了开放系统互连参考模型（OSI /RM）。OSI /RM 为开放系统互连提供了一种功能结构的框架。OSI /RM 是一种分层的体系结构，参考模型共有 7 层，分层的基本想法是每一层都在它的下层提供的服务基础上提供更高级的增值服务，而最高层提供能运行分布式应用程序的服务。这样，分层的方法就把复杂的问题分解开了，并保持了层次之间的独立性。OSI /RM 的网络体系结构如图 2-14 所示，由低层至高层分别为物理层（physical layer）、数据链路层（datalink layer）、网络层（network layer）、传输层（transport layer）、会话层（session layer）、表示层（presentation layer）和应用层（application layer）。

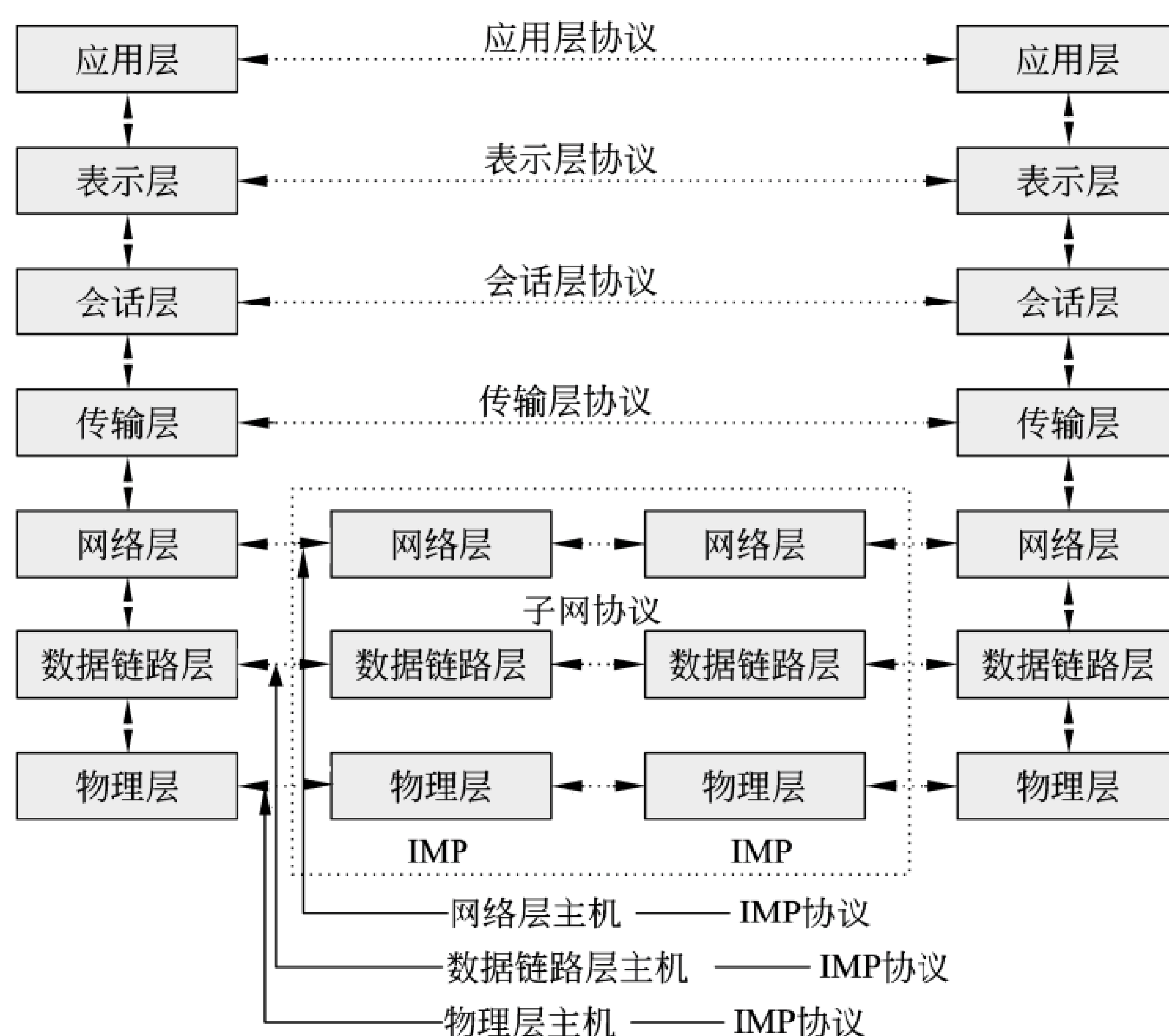


图 2-14　OSI 模型的网络体系结构

2．OSI 协议集

国际标准化组织除定义了开放系统互连（Open System Interconnection，OSI）参考模型之外，还开发了实现 7 个功能层次的各种协议和服务标准，这些协议和服务通称为“OSI 协议”。和其他协议一样，OSI 协议是实现某些功能的过程的描述和说明。每一个 OSI 协议都详细地规定了特定层次的功能特性。OSI 协议集如表 2-2 所示。

表 2-2　OSI 协议集

<table>
<tr><td rowspan="2">应用层</td><td>VT</td><td>DS</td><td>FTMA</td><td>CNIP/CMIS</td><td>MHS</td><td rowspan="2">ANS.1</td></tr>
<tr><td colspan="5">ACSE，RTSE，ROSE，CCR</td></tr>
<tr><td>表示层</td><td colspan="5">OSI 表示层协议</td><td></td></tr>
<tr><td>会话层</td><td colspan="5">OSI 会话层协议</td><td></td></tr>
<tr><td>传输层</td><td colspan="6">TP0，TP1，TP2，TP3，TP4</td></tr>
<tr><td rowspan="2">网络层</td><td colspan="6">ES-IS，IS-IS</td></tr>
<tr><td colspan="2">X.25 PLP</td><td colspan="4">CLNP</td></tr>
<tr><td>数据链路层</td><td colspan="2">IEEE 802.2</td><td colspan="4">HDLC　PAP-B</td></tr>
<tr><td>物理层</td><td colspan="2">802.3　802.4　802.5　FDDI</td><td colspan="4">RS-232 RS-449 X-21 V.35 ISDN</td></tr>
</table>

2.3.2　计算机网络

1．广域网、局域网和城域网

广域网又称远程网，它是指覆盖范围广，传输速率相对较低，以数据通信为主要目的的数据通信网。它的特点是：分布范围广，数据传输率低，数据传输可靠性随着传输介质的不同而不同、拓扑结构复杂。广域通信网有公共交换电话网和各种公用数据网（包括分组交换网、帧中继网、ATM 网、移动通信网等）。目前用于广域传输的协议有 PPP（点对点协议）、DDN、ISDN（综合业务数字网）、FR（帧中继）和 ATM（异步传输模式）等。

局域网是指传输距离有限，传输速度较高，以共享网络资源为目的的网络系统，它的特点是：分布范围有限，有较高的通信带宽和数据传输率高，数据传输可靠误码率低，通常采用同轴电缆或双绞线作为传输介质，拓扑结构简单简洁，网络的控制一般为分布式，通常被单一组织所拥有和使用。局域网使用的拓扑结构有总线拓扑、环型拓扑、星形拓扑以及它们的混合型。

城域网是规模介于局域网和广域网之间的一种较大范围的高速网络，一般覆盖临近的多个单位和城市，从而为接入网络的企业、机关、公司及社会单位提供文字、声音和图像的集成服务。

网络拓扑结构是指网络中通信线路和节点的几何排序，用以表示整个网络的结构外貌，反映各节点之间的结构关系。它影响着整个网络的设计、功能、可靠性和通信费用等重要方面。常用的网络拓扑结构有总线型、星型、环型、树型和分布式结构等。

局域网和城域网的国际标准都是 IEEE802 标准。决定局域网的主要技术有用以传输数据的传输介质、用以连接各种设备的拓扑结构、用以共享资源的介质访问控制方法。这三种技术在很大程度上决定了传输数据的类型、网络的响应时间、吞吐率和利用率，以及网络应用等各种网络特性。其中最重要的是介质访问控制方法。在局域网和城域网

中，所有的设备都共享传输介质，所以需要一种方法能有效地分配传输介质的使用权，这种功能就叫做介质访问控制协议。对总线型、星型和树型拓扑结构最适合的介质访问控制协议是 CSMA/CD（Carrier Sense Multiple Access/Collision Detection）介质访问控制方法有集中式控制和分布式控制两种。

国际电子电气工程师协会 IEEE 制定的局域网的标准：IEEE 802.3（CSMA / CD，以太网）、IEEE 802.4（Token Bus，令牌总线）、IEEE 802.5（Token Ring，令牌环），由于它们已被市场广泛接受，所以 IEEE 802 系列标准已被 ISO 采纳为国际标准。随着网络技术的发展，又出现了 IEEE 802.7（FDDI）、IEEE 802.3u（快速以太网）、IEEE 802.12（100VG-AnyLAN）、IEEE 802.3z（千兆以太网）等新一代网络标准。

无线局域网（Wireless Local Area Networks，WLAN）就是在不采用传统缆线的同时，提供以太网或者令牌网络的功能。与有线网络相比，无线局域网具有以下优点：安装便捷，使用灵活，经济节约，易于扩展。IEEE 802.11 标准是由面向数据的计算机局域网发展而来，网络采用无连接的协议，目前市场上的大部分产品都是根据这个标准开发的。无线局域网可以在普通局域网基础上通过无线 Hub、无线接入站（AP）、无线网桥、无线 Modem 及无线网卡等来实现，其中以无线网卡最为普遍，使用最多。无线局域网的关键技术，除了红外传输技术、扩频技术、窄带微波技术外还有一些其他技术，如调制技术、加解扰技术、无线分集接收技术、功率控制技术和节能技术。无线局域网在室外主要有以下几种结构：点对点型、点对多点型、多点对点型和混合型。基于无线局域网具有的诸多优点，它可广泛应用于下列领域：接入网络信息系统、难以布线的环境、频繁变化的环境、使用便携式计算机等可移动设备进行快速网络连接、用于远距离信息的传输、专门工程或高峰时间所需的暂时局域网、流动工作者可得到信息的区域、办公室和家庭办公室（Small office/Home office，SOHO）用户，以及需要方便快捷地安装小型网络的用户。

2．网络互联

网络互连目的是使一个网络的用户能访问其他网络的资源，使不同网络上用户能够互相通信和交换信息，实现更大范围的资源共享。在网络互连时，一般不能简单地直接相连而通过一个中间设备来实现。

网络互联设备的作用是连接不同的网络，网络互联设备可以根据它们工作的协议层进行分类：中继器（repeater）、网桥（bridge）、路由器（router）、网关（gateway）和交换机等。在实际的网络互联产品中可能是几种功能的组合，从而可以提供更复杂的互联网服务。局域网用网桥互连，广域网的互联设备是路由器。网络线路与用户节点连接时需要的是网络传输介质互连设备，如 T 型头、收发器、RJ-45、RS232 接口、DB-15 接口、VB35 同步接口、网络接口单元和调制解调器等。物理层的互连设备有中继器和集线器（hub），数据链路层的互连设备有网桥、交换机，网络层互连设备是路由器，网关是应用

层互连设备。

传输介质是信号传输的媒体，常用的介质分为有线介质和无线介质。有线介质有双绞线、同轴电缆和光纤等；无线介质有微波、红外线和激光等。在一个局域网中，其基本组成部件为服务器、客户机、网络设备、通信介质和网络软件等。

3．Internet 及应用

Internet，又称因特网，是世界上规模最大、覆盖面最广且最具影响力的计算机互联网络，它将分布在世界各地的计算机利用开放系统互联协议连接在一起，用来进行数据传输、信息交换和资源共享。

用户接入因特网的方式有终端方式、SLIP/PPP 方式、专线方式（DDN、FR、ISDN 专线、网络电缆直连）、代理服务器方式等。

TCP/IP（Transmission Control Protocol /Internet Protocol）作为 Internet 的核心协议，已被广泛应用于局域网和广域网中，TCP/IP 的主要特性为逻辑编址、路由选择、域名解析、错误检测和流量控制以及对应用程序的支持等。

TCP/IP 是个协议族，它包含多种协议。ISO/OSI 模型与 TCP/IP 模型的对比如表 2-3 所示。

表 2-3　TCP/IP 模型与 OSI 模型的对比

ISO/OSI 模型	TCP/IP 协议					TCP/IP 模型
应用层	文件传输协议 FTP	远程登录协议 Telnet	电子邮件协议 SMTP	网络文件服务协议 NFS	网络管理协议 SNMP	应用层
表示层						
会话层						
传输层	TCP			UDP		传输层
网络层	IP	ICMP		ARP	RARP	网际层
数据链路层	Enternet IEEE 802.3	FDDI	Token-Ring/IEEE 802.3	ARCnet	PPP/SLIP	网络接口层
物理层						硬件层

从表上可知，TCP/IP 分层模型由 4 个层次构成，即应用层、传输层、网际层和网络接口层。网际层定义的协议除了 IP 外，还有（Internet Control Message Protocol，ICMP）、（Address Resolution Protocol，ARP）和（Reverse Address Resolution Protocol，RARP）等几个重要协议。应用层的协议有（Network File Serve，NFS）、Telnet、（Simple Mail Transport Protocol，SMTP）、（Simple Network Management Protocol，SNMP）和（File Transfer Protocol，FTP）等。

Internet 的地址主要有两种书写形式：域名格式和 IP 地址格式。域名和 IP 地址是一一对应的。现在的 IP 协议版本号为 4，也称之为 IPv4，新的 IP 协议为 IPv6，IPv6 将彻

底解决 IP 地址缺乏问题。

WWW（World Wide Web），也称万维网或全球网，是指在因特网上以超文本为基础形成的信息网。它采用统一的资源定位器（Uniform Resource Locator，URL）和图文声并茂的用户界面，可以方便地浏览 Internet 上的信息和利用各种网络服务。互联网常用的服务包括：域名服务（Domain Name Server，DNS）、WWW 服务、E-mail 电子邮件服务、FTP 文件传输服务、Telnet 远程登录服务、Gopher 等等。

2.3.3 网络管理与网络安全

1．网络管理

网络管理是对计算机网络的配置、运行状态和计费等进行的管理。它提供了监控、协调和测试各种网络资源以及网络运行状况的手段，还可提供安全字处理和计费等功能。在 OSI 网络管理标准中定义了网络管理的五大基本功能：配置管理、性能管理、故障管理、安全管理和计费管理。事实上，网络管理还应该包括其他一些功能，如网络规划、网络操作人员的管理等。

2．计算机网络安全

计算机网络安全是指计算机、网络系统的硬件、软件以及系统中的数据受到保护，不因偶然的或恶意的原因而遭到破坏、更改、泄露，确保系统能连续和可靠地运行，使网络服务不中断。网络安全从本质上讲就是网络上的信息安全。信息安全是在分布式计算环境中对信息的传输、存储、访问提供安全保护，以防止信息被窃取、篡改和非法操作。信息安全的基本要素是保密性、完整性、可用性、真实性和可控性。完整的信息安全保障体系应包括保护、检测、响应和恢复等 4 个方面。信息安全术语有密码学、鉴别、Kerberos 鉴别、公钥基础设施、数字签名、访问控制。

网络威胁：是对网络安全缺陷的潜在利用，这些缺陷可能导致非授权访问、信息泄露、资源耗尽、资源被盗或者被破坏等。网络安全威胁的种类有窃听、假冒、重放、流量分析、数据完整性破坏、拒绝服务、资源的非授权使用、陷门和特洛伊木马、病毒、诽谤等。

网络安全漏洞：通常，入侵者首先寻找网络存在的安全弱点，然后从缺口处无声无息地进入网络。因而开发黑客反击武器的思想是找出现行网络中的安全弱点，演示、测试这些安全漏洞，然后指出应如何堵住安全漏洞。当前，信息系统的安全性非常脆弱，主要体现在操作系统、计算机网络和数据库管理系统都存在安全隐患，这些安全隐患表现在：物理安全性、软件安全漏洞、不兼容使用安全漏洞、选择合适的安全哲理。

网络攻击是指任何的非授权行为。攻击的范围从简单的使服务器无法提供正常的服务到完全破坏或控制服务器。在网络上成功实施的攻击级别依赖于用户采取的安全措施。

网络攻击有被动攻击、主动攻击、物理临近攻击、内部人员攻击和分发攻击。

任何形式的互联网络服务都会导致安全方面的风险，问题是如何将风险降低到最低程序。目前的网络安全措施有数据加密、数字签名、身份认证、防火墙和入侵检测等。

3．VPN

所谓虚拟专用网（Virtual Private Network，VPN）是建立在公用网上的、由某一组织或某一群用户专用的通信网络，其虚拟性表现在任意一对 VPN 用户之间没有专用的物理连接，而是通过（Internet Services Provider，ISP）提供的公用网络来实现通信；其专用性表现在 VPN 之外的用户无法访问 VPN 内部的网络资源，VPN 内部用户之间可以实现安全通信。这里讲的 VPN 是指在 Internet 上建立的、由用户（组织或个人）自行管理的 VPN，而不涉及一般电信网中的 VPN。

实现 VPN 的关键技术有隧道技术（Tunneling）、加解密技术（Encryption & Decryption）、密钥管理技术（Key Management）和身份认证技术（Authentication）。

VPN 的解决方案有三种：内联网 VPN（Intranet VPN）、外联网 VPN（Extranet VPN）和远程接入 VPN（Access VPN）。

2.3.4　网络工程

网络工程是根据用户单位的需求及具体情况，结合现时网络技术的发展水平及产品化程序，经过充分需求分析和市场调研，从而确定网络建设方案，依据方案有步骤、有计划实施的网络建设活动。网络工程建设是一项复杂的系统工程，一般可分为网络规划和网络设计阶段、工程组织和实施阶段以及系统运行维护阶段。

2.3.5　存储及负载均衡技术

1．RAID 技术

RAID（Redundant Array of Inexpensive Disks，磁盘阵列）是一种由多块廉价磁盘构成的冗余阵列。使用磁盘阵列的目的是建立数据冗余、增强容错、提高容量、增进性能。

RAID 技术主要包含 RAID 0～RAID 7 等规范，以及复合 RAID 模式 RAID 0+1、5+1 等。在 RAID 家族里，RAID 0 和 RAID 1 在个人电脑上得到了广泛的应用。

硬件 RAID 的实现：一般使用 SCSI 或者 IDE/ATA 作为硬盘系统的接口。硬件 RAID 实现分为两种：一种是内置（或集成）RAID 控制器，一种是外置 RAID 控制器。

软件 RAID 的实现：除了使用 RAID 卡或者主板所带的芯片实现磁盘阵列外，在一些操作系统中可以直接利用软件方式实现 RAID 功能，例如在 Windows 2000/XP 中就已经内置了 RAID 功能，Linux 用 Raidtools 来实现 RAID 功能。

2．网络存储技术

网络存储采用面向网络的存储体系结构，使数据处理和数据存储分离，由专门的系统负责数据处理，存储设备或子系统负责数据的存储。网络存储结构通过网络连接服务器和存储资源，具有灵活的网络寻址能力和远距离数据传输能力，实现了在一个或多个位置简单而可靠的数据存储、恢复和不同主机不同存储设备之间的资源共享。

网络存储体系结构大致分为三种：直连式存储（Direct Attached Storage，DAS）、网络连接存储（Network Attached Storage，NAS）和存储区域存储（Storage Area Network，SAN）。

3．负载均衡技术

负载均衡（Load Balance）是由多台服务器以对称的方式组成一个服务器集合，每台服务器都具有等价的地位，都可以单独对外提供服务而无须其他服务器的辅助。

负载均衡是在现有的网络结构的基础上，通过扩展网络设备和服务器的带宽，来增加吞吐量，提升网络的数据处理能力，提高网络的灵活性、可靠性、可用性和可维护性，最终目的是加快服务器的响应速度，从而提高用户的体验度。

负载均衡从结构上分为本地负载均衡（Local Server Load Balance）和全局负载均衡（Global Server Load Balance）。

负载均衡的实现方法有两种：第一种方法是把大量的并发访问或数据流量分配到多个设备上分别处理，以减少用户等待响应的时间；第二种方法是将单个的重负载的运算分摊到多个设备上做并行处理，再将每个设备的运行结果汇总后返回给用户。

一个网络的负载均衡，一般情况下从传输链路聚合、采用更高层网络交换技术和设置服务器群集策略三个角度来实现。常用的负载均衡技术有操作系统自带的负载均衡服务、基于特定服务器软件的负载均衡、基于 DNS 的负载均衡、反向代理负载均衡、基于 NAT 的负载均衡技术、扩展的负载均衡技术以及硬件方式。

4．服务器集群技术

集群（Cluster）是一组相互独立的服务器在网络中表现为单一的系统，并以单一系统的模式加以管理。此单一系统为客户端提供高可靠性的服务，并大幅提高了服务器的安全性。

一个 Cluster 包含多台（至少二台）拥有共享数据存储空间的服务器，任何一台服务器在运行一个应用时，应用数据被存储在共享的数据空间内。每台服务器的操作系统和应用程序文件存储在各自的本地存储空间上。

大多数模式下，集群中所有的计算机拥有一个共同的名称，各节点服务器通过一内部局域网相互通讯，集群内任一系统上运行的服务都可被所有的网络客户所使用，当一台节点服务器发生故障时，这台服务器上所运行的应用程序将在另一节点服务器上被自动接管，客户也能很快地自动地连接到新的应用服务器上。

集群服务在部署关键业务、电子商务、商务流程中的作用将日益重要起来。

2.4　多媒体技术及其应用

2.4.1　多媒体技术基本概念

1．媒体

媒体是指承载信息的载体，又称媒介。媒体有两种含义：一是表示信息的载体，如文本、图形、图像、动画、音频和视频等；二是存储信息的实体，如纸张、磁盘、光盘和半导体存储器等。

媒体的种类，根据 ITU-T（原 CCITT）建议的定义，媒体有 5 种：感觉媒体、表示媒体、显示媒体、存储媒体和传输媒体。

感觉媒体（Perception Medium）是指人们接触信息的感觉形式，如视觉、听觉、触觉、嗅觉和味觉等。

表示媒体（Representation Medium）是指信息的表示形式，如文字、图形、图像、动画、音频和视频等。

显示媒体（Presentation Medium）是表现和获取信息的物理设备。如输入显示媒体键盘、鼠标器和麦克风等；输出显示媒体显示器、打印机和音箱等。

存储媒体（Storage Medium）是存储数据的物理设备，如磁盘、光盘和内存等。

传输媒体（Transmission Medium）是指传输数据的物理载体，如电缆、光缆和交换设备等。

2．多媒体

多媒体是数字、文字、声音、图形、图像和动画等各种媒体的有机组合，并与先进的计算机、通信和广播电视技术相结合，形成一个可组织、存储、操纵和控制多媒体信息的集成环境和交互系统。由此可见，“多媒体”这个术语既指信息表示媒体的多样化，又包括了传播、处理和使用多媒体的各种技术和方法。

3．多媒体技术

多媒体技术是指以数字化为基础，能够对多种媒体信息进行采集、编码、存储、传输、处理和表现，综合处理多种媒体信息并使之建立起有机的逻辑联系，集成为一个系统并能具有良好交互性的技术。多媒体媒体元素是指多媒体应用中可显示给用户的媒体形式。目前我们常见的媒体元素主要有文本、图形、图像、声音、动画和视频图像等。多媒体技术包括计算机技术、视听技术及通信技术。

多媒体技术的特征有多样性、集成性、交互性和实时性。

4．多媒体计算机

多媒体计算机（Multimedia Personal Computer，MPC）是指能够综合处理多种媒体信息的计算机。也即：

MPC＝PC＋CD-ROM＋声卡＋显示卡＋多媒体操作系统

2.4.2 多媒体数据压缩编码技术

由国际标准化协会、国际电信协会和国际电联领导下，制定的三个有关视频图像压缩编码的国家标准：JPEG 标准、H·261 标准和 MPEG 标准。

1．多媒体数据压缩编码的国际标准

1）静态图像压缩编码的国际标准

JPEG（Joint Photographic Experts Group，联合图像专家小组标准）是一种对静态图像压缩的编码算法。“联合”的含义是：国际电报电话咨询委员会（Consultative Committee on International Telephone and Telegraph，CCITT）和国际标准化协会联合组成的图像专家小组。静态图像压缩标准有 JPEG、JPEG2000。

2）运动图像压缩标准

MPEG（Moving Picture Experts Group，运动图像专家组）是专门制定多媒体领域内的国际标准的一个组织，该组织成立于 1988 年，由全世界大约 300 名多媒体技术专家组成。MPEG 标准是面向运动图像压缩的一个系列标准。目前有 MPEG-1、MPEG-2、MPEG-4、MPEG-7、MPEG-21、DVI。

MPEG 的优势：首先，它是作为一个国际化的标准来研究制定的，所以，具有很好的兼容性。其次，MPEG 能够比其他算法提供更好的压缩比，最高可达 200:1。更重要的是，MPEG 在提供高压缩比的同时，对数据的损失很小。与同样是音频压缩标准的 AC 系列标准相比，MPEG 标准系列由于不存在专利权的问题，它更适合于大力推广。随着 MPEG 新标准的不断推出，数据压缩和传输技术必将趋向更加规范化。

2．多媒体数据压缩方法的分类

目前常用的数据压缩编码方法可以分为两大类：一类是无损压缩编码法（Lossless Compression Coding），也称冗余压缩法或熵编码法；另一类是有损压缩编码法（Loss Compression Coding），也称为熵压缩法，如图 2-15 所示。

2.4.3 多媒体系统的组成

多媒体系统的层次结构与计算机系统的结构在原则上是相同的，由底层的硬件系统和其上的各层软件系统组成，只是考虑多媒体的特性各层次的内容有所不同。多媒体系统的层次结构如图 2-16 所示。

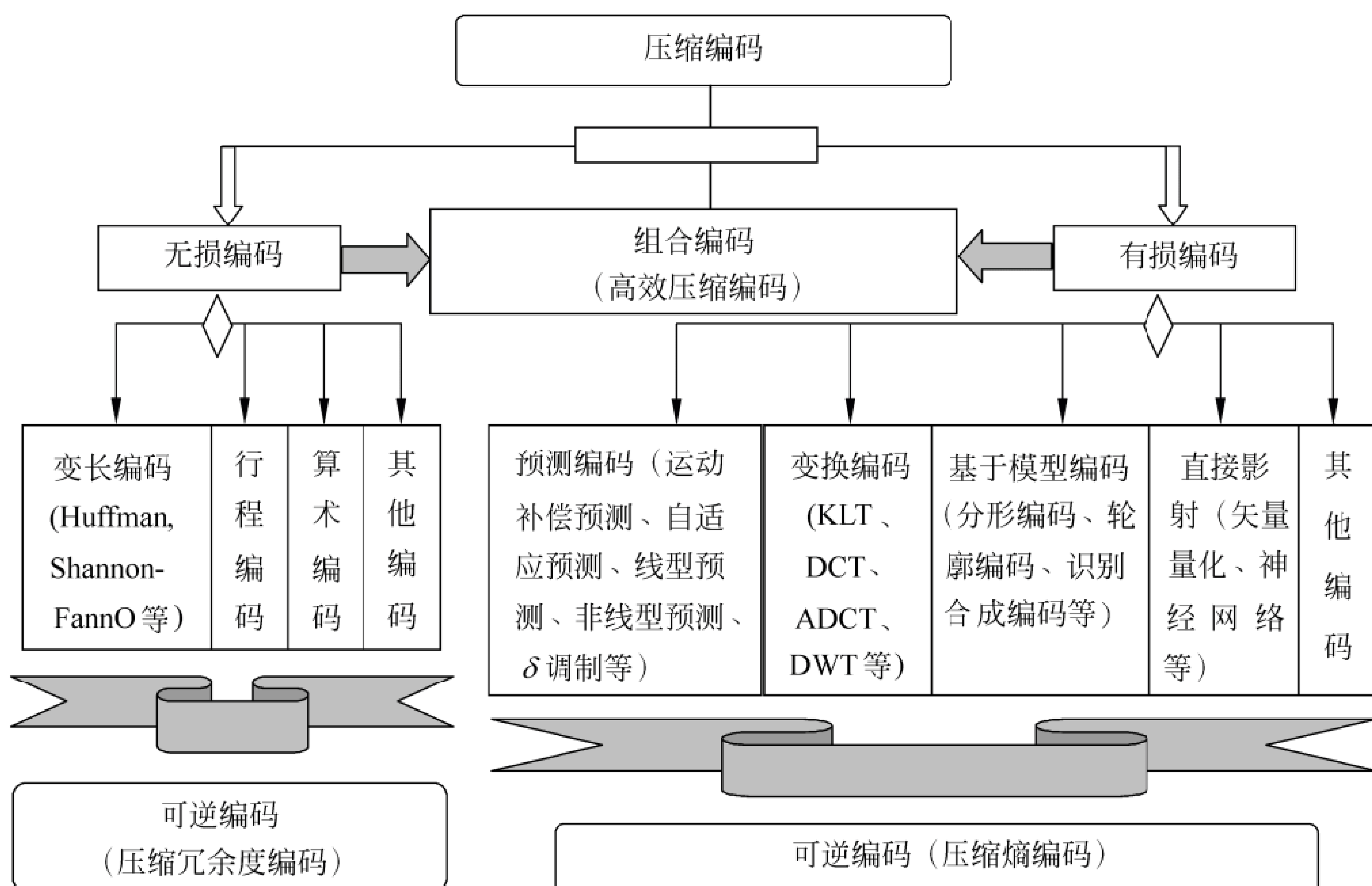


图 2-15　常用压缩编码方法分类

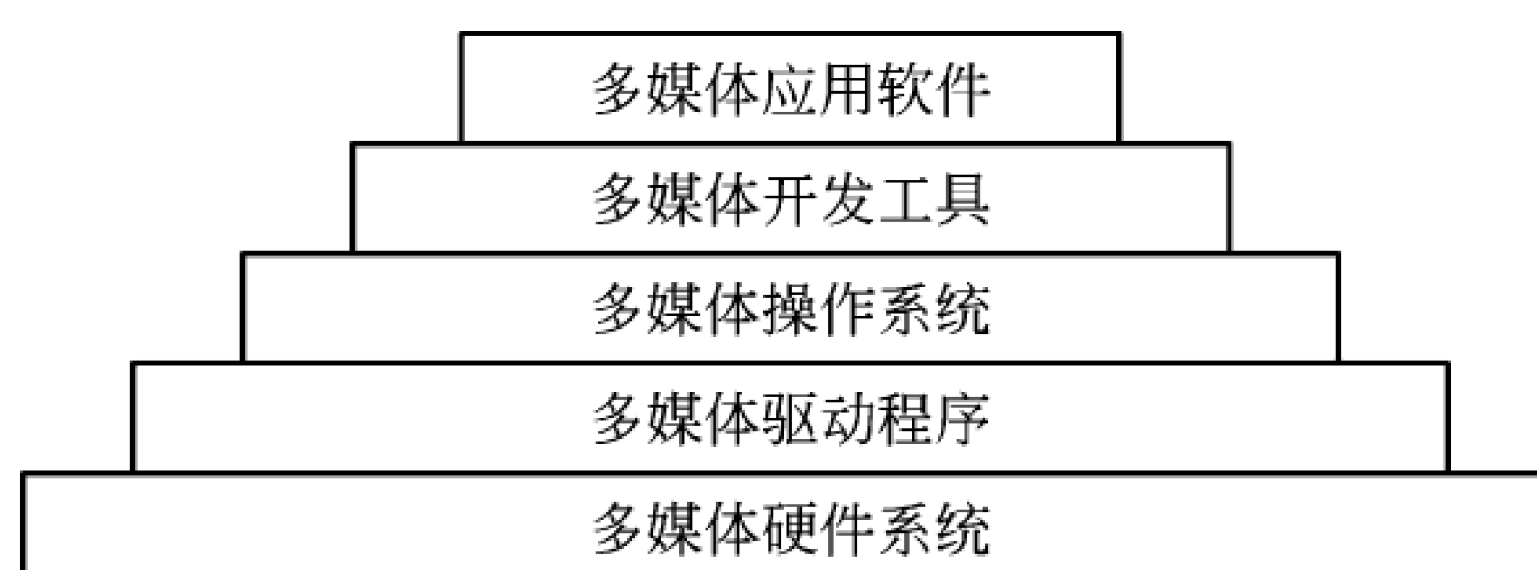


图 2-16 多媒体系统的层次结构

1．多媒体硬件

多媒体硬件系统由主机、多媒体外部设备接口卡和多媒体外部设备构成。

多媒体计算机的主机可以是大/中型计算机，也可以是工作站，用得最多的是微机。

多媒体外部设备接口卡根据获取、编辑音频、视频的需要插接在计算机上。常用的有声卡、视频压缩卡、VGA/TV 转换卡、视频捕捉卡、视频播放卡和光盘接口卡等。

多媒体外部设备十分丰富，按功能分为视频/音频输入设备、视频/音频输出设备、人机交互设备、数据存储设备 4 类。视频/音频输入设备包括摄像机、录像机、影碟机、扫描仪、话筒、录音机、激光唱盘和 MIDI 合成器等；视频/音频输出设备包括显示器、电视机、投影电视、扬声器和立体声耳机等；人机交互设备包括键盘、鼠标、触摸屏和光笔等；数据存储设备包括 CD-ROM、磁盘、打印机和可擦写光盘等。

2．多媒体软件

多媒体软件系统按功能可分为系统软件和应用软件。

系统软件是多媒体系统的核心，它不仅具有综合使用各种媒体、灵活调度多媒体数据进行媒体的传输和处理的能力，而且要控制各种媒体硬件设备协调地工作。多媒体系统软件主要包括多媒体操作系统、媒体素材制作软件及多媒体函数库、多媒体创作工具与开发环境、多媒体外部设备驱动软件和驱动器接口程序等，如下。

- 多媒体素材制作工具软件。
- 文字特效制作软件：MSWord（艺术字）、UleadCOOL3D。
- 音频处理软件：Syntrillium Cooledit、TwelveTone Cakewalk。
- 图形与图像处理软件：CorelDRAW、Adobe Photoshop、Ulead PhotoImpact。
- 动画制作软件：Macromedia FlashMX、Discreet 3dsmax、Alias/Wavefront Maya。
- 视频编辑软件：AdobePremiere、UleadMediaStudio。
- 多媒体著作工具软件：Microsoft PowerPoint、Microsoft FrontPage、Macromedia Authorware、Macromedia Director。
- 多媒体编程语言：VB、VC++、Delphi。

应用软件是在多媒体创作平台上设计开发的面向应用领域的软件系统，通常由应用领域的专家和多媒体开发人员共同协作、配合完成的多媒体应用系统和多媒体产品。例如，教育软件、电子图书等。

2.4.4 多媒体技术的研究内容

1．数据压缩

在多媒体系统中，由于涉及的各种媒体信息主要是非常规数据类型，如图形、图像、视频和音频等，这些数据所需要的存储空间是十分巨大和惊人的。为了使多媒体技术达到实用水平，除了采用新技术手段增加存储空间和通信带宽外，对数据进行有效压缩将是多媒体发展中必须要解决的最关键的技术之一。

2．数据的组织与管理

数据量大，种类繁多，关系复杂是多媒体数据的基本特征。面向对象数据库（Object Oriented Data Base，OODB）和多媒体数据库结合超媒体（hypermedia）技术的应用，为多媒体信息的建模、组织和管理提供了有效的方法。

3．多媒体信息的展现与交互

在多媒体环境下，各种媒体并存，视觉、听觉、触觉、味觉和嗅觉媒体信息的综合与合成，各种媒体的时空安排和效应，相互之间的同步和合成效果，相互作用的解释和描述等都是多媒体领域需要研究和解决的问题。

4．多媒体通信与分布处理

多媒体通信对多媒体产业的发展、普及和应用有着举足轻重的作用，构成了整个产

业发展的关键和瓶颈。还需要优化现有的电话网、广播电视网和计算机网络，以使其传输性能能较好地满足多媒体数据数字化通信的需求。

多媒体的分布处理是一个十分重要的研究课题。因为要想广泛地实现信息共享，计算机网及其在网络上的分布式与协作操作就不可避免。多媒体空间的合理分布和有效的协作操作将缩小个体与群体、局部与全球的工作差距。超越时空限制，充分利用信息，协同合作，相互交流，节约时间和经费等是多媒体信息分布的基本目标。

5. 虚拟现实技术

所谓虚拟现实，就是采用计算机技术生成一个逼真的视觉、听觉、触觉及味觉等感官世界，用户可以直接用人的技能和智慧对这个生成的虚拟实体进行考察和操纵。这个概念包含三层含义：首先，虚拟现实是用计算机生成的一个逼真的实体，“逼真”就是要达到三维视觉、听觉和触觉等效果；其次，用户可以通过人的感官与这个环境进行交互；最后，虚拟现实往往要借助一些三维传感技术为用户提供一个逼真的操作环境。

虚拟现实是一种多技术多学科相互渗透和集成的技术，研究难度非常大。但由于它是多媒体应用的高级境界，且应用前景远大，而且某些方面的应用甚至远远地超过了这种技术本身的研究价值。

6. 智能多媒体技术

将具有推理功能的知识库与多媒体数据库结合起来，形成智能多媒体数据库。智能多媒体数据库另一个重要研究课题是多媒体数据库基于内容检索技术，它需要把人工智能领域中的高维空间的搜索技术、视音频信息的特征抽取和识别技术、视音频信息的语义抽取问题以及知识工程中的学习、挖掘及推理等问题应用到基于内容的检索技术中。

7. 把多媒体信息实时处理和压缩编码算法集成到 CPU 芯片中

计算机产业的发展趋势应该是把多媒体和通讯的功能集成到 CPU 芯片中，过去计算机结构设计较多地考虑计算功能，主要用于数学运算及数值处理，随着多媒体技术和网络通讯技术的发展，需要计算机具有综合处理声、文、图信息及通讯的功能。其一，是以多媒体和通讯功能为主，融合 CPU 芯片原有的计算功能，它的设计目标是用在多媒体专用设备、家电及宽带通讯设备，可以取代这些设备中的 CPU 及大量 Asic 和其他芯片。其二，是以通用 CPU 计算功能为主，融合多媒体和通讯功能，它们的设计目标是与现有的计算机系列兼容，同时具有多媒体和通讯功能，主要用在多媒体计算机中。

2.4.5　多媒体技术的应用领域

多媒体技术为计算机的应用开拓了更广阔的领域，不仅涉及到计算机的各个应用领域，也涉及到通信、传播、出版、商业广告及购物、文化娱乐、工程设计等各种领域或行业。多媒体在各行各业领域中的应用又推动了多媒体技术与产品的发展，开创了多媒体技术发展的新时代。各种计算机应用软件都竞相加入多媒体元素，多媒体节目也渗入到各行各业中，并进入到人们的家庭生活和娱乐中。

1．办公自动化

多媒体技术的出现为办公室增加了控制信息的能力和充分表达思想的机会，许多应用程序都是为提高办公人员的工作效率而设计的，从而产生了许多新型的办公自动化系统。由于采用了先进的数字影像和多媒体计算机技术，把文件扫描仪、图文传真机、文件资料微缩系统和通信网络等现代化办公设备综合管理起来，构成了全新的办公自动化系统。

2．电子出版物

电子出版物是指以数字代码方式将图、文、声、像等信息存储在磁、光、电介质上，通过计算机或类似设备阅读使用，并可复制发行的大众传播媒体。电子出版物的内容可分为电子图书、手册、文档、报刊杂志、教育培训、娱乐游戏、宣传广告、信息咨询和简报等，许多作品是多种类型的组合。多媒体电子出版物是计算机多媒体技术与文化、文艺、教育等多种学科相结合的产物。

3．多媒体通信

随着网络的发展，电子邮件已被普遍采用。而包括声、文、图在内的多媒体邮件更受到用户的普遍欢迎，在此基础上发展起来的可视电话、视频会议系统、数字家电（电话、电视、传真、音响）和远程医疗系统为人类提供了全新的服务方式。

多媒体通信有着极其广泛的内容，信息点播（Information Demand）和计算机协同工作（Computer Supported Cooperative Work，CSCW）系统对人类生活、学习和工作产生了深刻的影响。

4．教育与培训

以多媒体计算机为核心的现代教育技术使教学手段和方法丰富多彩，使计算机教学如虎添翼。多媒体教学不仅使学生获得生动的学习环境，而且使教师拥有高水平、高质量的教学环境。

正是因为多媒体教育对于促进教学思想、教学内容和教学手段的改革，实现多元化、主体化和社会化，全面提高教学质量有着重大的意义，网络课程、虚拟课堂、虚拟实验室、数字图书馆、多媒体技能培训系统等多媒体教育产品已广泛用于初、中级基础教育，高等教育及职业培训等方面。

5．商业与咨询

各类商家将各种服务指南存放于多媒体系统中向公众展示、推介和咨询的有多媒体商业简报、产品演示、查询服务等。

6．军事与娱乐

将多媒体技术应用于军事和娱乐的有军事遥感、核武器模拟、战场模拟、CD、MIDI、VCD、DVD、游戏等。

不难看出，多媒体技术在人类工作、学习、信息服务、娱乐、家庭生活及艺术创作等各个领域都表现出非凡的能力，并在不断开拓新的应用领域。

2.5　系统性能

系统性能是一个系统提供给用户的众多性能指标的集合。它既包括硬件性能，也包括软件性能；既包括部件性能指标，也包括综合性能指标。系统性能包含性能指标、性能计算、性能设计和性能评估 4 个方面的内容。

2.5.1　性能指标

性能指标，是软、硬件的性能指标的集成。在硬件中，包括计算机、各种通信交换设备、各类网络设备等；在软件中，包括操作系统、协议以及应用程序等。

1．计算机

对计算机评价的主要性能指标有时钟频率（主频）、运算速度、运算精度、内存的存储容量、存储器的存取周期、数据处理速率（Processing Data Rate，PDR）；吞吐率、各种响应时间、各种利用率、RASIS 特性，即可靠性（Reliability）、可用性（Availability）、可维护性（Sericeability）、完整性和安全性（Integraity and Security）；平均故障响应时间、兼容性、可扩充性、性能价格比。

2．路由器

对路由器评价的主要性能指标有设备吞吐量、端口吞吐量、全双工线速转发能力、背靠背帧数、路由表能力、背板能力、丢包率、时延、时延抖动、VPN 支持能力、内部时钟精度、队列管理机制、端口硬件队列数、分类业务带宽保证、RSVP、IP Diff Serv、CAR 支持、冗余、热插拔组件、路由器冗余协议、网管、基于 Web 的管理、网管类型、带外网管支持、网管粒度、计费能力 / 协议、分组语音支持方式、协议支持、语音压缩能力、端口密度、信令支持。

3．交换机

对交换机评价所依据的性能有交换机类型、配置、支持的网络类型、最大 ATM 端口数、最大 SONET 端口数、最大 FDDI 端口数、背板吞吐量、缓冲区大小、最大 MAC 地址表大小、最大电源数、支持协议和标准、路由信息协议（RIP）、RIP2、开放式最短路径优先第 2 版、边界网关协议（BGP）、无类别域间路由（CIDR）、互联网成组管理协议（IGMP）、距离矢量多播路由协议（DVMRP）、开放式最短路径优先多播路由协议（MOSPF）、协议无关的多播协议（PIM）、资源预留协议（RSVP）、802.1p 优先级标记，多队列、路由、支持第 3 层交换、支持多层（4 到 7 层交换、支持多协议路由、支持路由缓存、可支持最大路由表数、VLAN、最大 VLAN 数量、网管、支持网管类型、支持端口镜像、QoS、支持基于策略的第 2 层交换、每端口最大优先级队列数、支持基于策略的第 3 层交换、支持基于策略的应用级 QoS、支持最小 / 最大带宽分配、冗余、热交

换组件（管理卡，交换结构，接口模块，电源，冷却系统、支持端口链路聚集协议、负载均衡。

4．网络

评价网络的性能指标有设备级性能指标、网络级性能指标、应用级性能指标、用户级性能指标、吞吐量。

5．操作系统

评价操作系统的性能指标有系统的可靠性、系统的吞吐率（量）、系统响应时间、系统资源利用率、可移植性。

6．数据库管理系统

衡量数据库管理系统的主要性能指标包括数据库本身和管理系统两部分，有数据库的大小、数据库中表的数量、单个表的大小、表中允许的记录（行）数量、单个记录（行）的大小、表上所允许的索引数量、数据库所允许的索引数量、最大并发事务处理能力、负载均衡能力、最大连接数等等。

7．Web 服务器

评价 Web 服务器的主要性能指标有最大并发连接数、响应延迟、吞吐量。

2.5.2 性能计算

性能指标计算的主要方法有定义法、公式法、程序检测法和仪器检测法。

常用的性能指标的计算过程（Millions of Instructions Per Second，MIPS）的计算方法、峰值计算、等效指令速度（吉普森（Gibson）法）。

在实际应用中，往往是对这些常用性能指标的复合计算，然后通过算法加权处理得到最终结果。

2.5.3 性能设计

1．性能调整

当系统性能降到最基本的水平时，性能调整由查找和消除瓶颈组成。对于数据库系统，性能调整主要包括 CPU / 内存使用状况、优化数据库设计、优化数据库管理以及进程/线程状态、硬盘剩余空间、日志文件大小等；对于应用系统，性能调整主要包括应用系统的可用性、响应时间、并发用户数以及特定应用的系统资源占用等。

在开始性能调整之前，必须做的准备工作有识别约束、指定负载、设置性能目标。在建立了性能调整的边界和期望值后，就可以开始调整了，这是一系列重复的受控的性能试验，循环的调整过程为收集、分析、配置和测试。

2．阿姆达尔解决方案

阿姆达尔（Amdahl）定律主要用于系统性能改进的计算中。阿姆达尔定律是指计算

机系统中对某一部件采用某种更快的执行方式所获得的系统性能改变程度，取决于这种方式被使用的频率，或所占总执行时间的比例。

阿姆达尔定律定义了采用特定部件所取得的加速比。假定我们使用某种增强部件，计算机的性能就会得到提高，那么加速比就是下式所定义的比率：

$$\text{加速比}=\frac{\text{不使用增强部件时完成整个任务的时间}}{\text{使用增强部件时完成整个任务的时间}}$$

加速比反映了使用增强部件后完成一个任务比不使用增强部件完成同一任务加快了多少。加速比主要取决于两个因素：

（1）在原有的计算机上，能被改进并增强的部分在总执行时间中所占的比例。这个值称为增强比例，它永远小于等于 1。

（2）通过增强的执行方式所取得的改进，即如果整个程序使用了增强的执行方式，那么这个任务的执行速度会有多少提高，这个值是在原来条件下程序的执行时间与使用增强功能后程序的执行时间之比。

原来的机器使用了增强功能后，执行时间等于未改进部分的执行时间加上改进部分的执行时间。

$$\text{新的执行时间}=\text{原来的执行时间}\times\left((\ -\text{增强比例})\quad\frac{\text{增强比例}}{\text{增强加速比}}\right)$$

总的加速比等于两种执行时间的比：

$$\text{总加速比}=\frac{\text{原来的执行时间}}{\text{新的执行时间}}=\frac{1}{\left((1-\text{增强比例})+\dfrac{\text{增强比例}}{\text{增强加速比}}\right)}$$

2.5.4　性能评估

性能评估是为了一个目的，按照一定的步骤，选用一定的度量项目，通过建模和实验，对一个系统的性能进行各项检测，对测试结果作出解释，并形成一份文档的技术。性能评估的一个目的是为性能的优化提供参考。

1．基准测试程序

大多数情况下，为测试新系统的性能，用户必须依靠评价程序来评价机器的性能。下面列出了 4 种评价程序，它们评测的准确程度依次递减：真实的程序、核心程序、小型基准程序、合成基准程序。

把应用程序中用得最多、最频繁的那部分核心程序作为评价计算机性能的标准程序，称为基准测试程序（benchmark）。基准测试程序有整数测试程序 Dhrystone、浮点测试程序 Linpack、Whetstone 基准测试程序、SPEC 基准测试程序和 TPC 基准程序。

2．Web 服务器的性能评估

在 Web 服务器的测试中，反映其性能的指标主要有：最大并发连接数、响应延迟和

吞吐量等。

常见的 Web 服务器性能评测方法有基准性能测试、压力测试和可靠性测试。

3．系统监视

进行系统监视通常有三种方式：一是通过系统本身提供的命令，如 UNIX/Linux 中的 W、ps、last，Windows 中的 netstat 等；二是通过系统记录文件查阅系统在特定时间内的运行状态；三是集成命令、文件记录和可视化技术，如 Windows 的 Perfmon 应用程序。

第 3 章　信息系统基础知识

3.1　信息化概述

3.1.1　信息的定义

随着现代通信技术的迅速发展与普及，信息的应用日益广泛，各种信息系统已经成为国家基础设施，支持着电子政务、电子商务、电子金融、科学研究、通信和社会保障等众多领域的发展，使人类继工业社会之后，正式迈入信息社会。信息的增长速度和利用程度，已成为现代社会文明和科技进步的重要标志。但是，理性认识信息却只有几十年的历史。

1928 年，哈补莱（L. v R. Hartlly）在《贝尔系统技术杂志》上发表了一篇题为“信息传输”的论文。在这篇论文中，他把信息理解为选择通信符号的方式，且用选择的自由度来计量这种信息的大小。1948 年，香农在《通信的数学理论》一文中把“信息”解释为“减少不确定性的东西”。由此引申出信息的一个定义：“信息是系统有序程度的度量”。同年，控制论的创始人维纳在《控制论》一书中指出：“信息就是信息，不是物质也不是能量”。1975 年，意大利学者朗高（G. Longo）在《信息沦：新的趋势与未决问题》一书的序言中认为“信息是反映事物的形式、关系相差别的东西，它包含在事物的差异之中，而不在事物本身”。以后人们还从不同的角度给信息下了定义，据统计，目前信息的定义不下几十种。目前，关于信息比较统一和科学的定义是系统论对信息的概括，即信息是对客观事物变化和特征的反映，是客观事物之间相互作用和联系的表征，是客观事物经过感知或认识后的再现。

3.1.2　信息的特征

信息具有以下特征。

（1）客观性：信息反映了客观事物的运动状态和方式。客观性也即事实性，不符合事实的信息不仅没有价值，而且可能有副作用。

（2）普遍性：物质的普遍性决定了信息的普遍存在，因而信息无所不在。

（3）无限性：由于一切事物运动的状态和方式都是信息，而事物及其变化是无限多样的，因而信息是无限的。

（4）动态性：信息是随着时间的变化而变化，因而是动态的。

（5）依附性：信息是客观世界的反映，因而要依附于一定的载体而存在，需要有物质的承担者。信息不能完全脱离物质而独立存在。

（6）变换性：信息是可变换的，它可以用不同的载体以不同的方法来负载。

（7）传递性：信息可以在时间上或空间上从一点传递到另一点。信息在时间上的传递即是存储，在空间上的传递即是转移或扩散。

（8）层次性：客观世界是分层次的，反映它的信息也是分层次的。信息可分为战略级、管理级和操作级。

（9）系统性：信息可以表示为一种集合，不同类别的信息可以形成不同的整体。因而，可以形成与现实世界相对应的信息系统。

3.1.3 信息化的定义

1963 年日本学者梅田忠夫首次提出了信息化的概念（Informationalization）。所谓信息化是指在国家宏观信息政策指导下，通过信息技术开发、信息产业的发展、信息人才的配置，最大限度地利用信息资源以满足全社会的信息需求，从而加速社会各个领域的共同发展以推进信息社会的过程。从本质上看，信息化应该是以信息资源开发利用为核心，以网络技术、通讯技术等高科技技术为依托的一种新技术扩散的过程。在信息化过程中，信息技术自身和整个社会都发生着质的变化。信息化不仅仅是生产力的变革，而且伴随着生产关系的重大变革。信息化的主体是全体社会成员，包括政府、企业、事业、团体和个人；它的时域是一个长期的过程；它的空域是经济和社会的一切领域；它的手段是基于现代信息技术的先进社会生产工具；它的途径是创建信息时代的先进生产力，推动社会生产关系及社会上层建筑的改革；它的目标是使国家的综合实力、社会的文明素质和人民的生活质量全面达到现代化水平。工业化、现代化和自动化都是信息化的基础；反过来，信息化则是工业化、现代化和自动化向高级阶段发展的必然结果。

3.1.4 信息化的内容

从信息化建设的角度出发，信息化的内容主要有 6 个要素，信息化的内容总是围绕着这 6 个要素展开的。

（1）信息资源的开发利用。信息化本来就是信息资源的大量开发和利用过程，因此信息资源是信息化源泉。信息资源的开发利用要解决三个问题：

① 原始信息的采集。

② 使存在的信息在给定的时间内获得。

③ 使用户获得真正需要的信息。

（2）信息网络的全面覆盖。信息网络是信息资源开发、利用的基础设施，是信息传输、交换和共享的必要手段。只有建设先进的信息网络，才能充分发挥信息化的整体效益。信息网络包括计算机网络、电信网和电视网等。信息网络在国家信息化的过程中将

逐步实现三网融合，并最终做到三网合一。

（3）信息技术的广泛应用。信息技术的应用是指把信息技术广泛应用于经济和社会各个领域，这是信息化的基础。信息技术应用是国家信息化中十分重要的要素，它直接反映了效率、效果和效益。

（4）信息产业的大力发展。信息产业是信息化的物质基础。信息产业包括微电子、计算机、电信等产品和技术的开发、生产、销售，以及软件、信息系统开发和电子商务等。从根本上来说，国家信息化只有在产品和技术方面拥有雄厚的自主知识产权，才能提高综合国力。

（5）信息化人才的培养。高素质的人才队伍和合理的人才结构是信息化建设能否取得成功的关键所在。合理的信息化人才结构要求不仅要有各个层次的信息化技术人才，还要有精干的信息化管理人才、营销人才、法律人才和情报人才。

（6）信息化政策法规和标准规范建设。信息化政策法规和标准规范是国家信息化快速、有序、健康和持续发展的保障，主要包括了电子商务交易、知识产权保护、信息资源管理、网络安全、信息管理和安全认证等法规标准。

3.1.5　信息化的经济社会意义

信息化的经济社会意义主要表现在以下几个方面。

（1）信息化促进全球化的发展。具体表现在以下 4 个方面：信息技术产品贸易直接促进世界商品贸易的增长；信息技术促进服务贸易的发展；信息技术为跨国公司的投资、贸易活动提供便利；信息技术促进金融全球化和全球金融市场的形成。

（2）信息化极大地促进了经济的增长。主要表现为两个方面：一是促进经济结构转变与产业结构高级化；二是信息和知识作为经济增长的内在因素，作为增长内在的源泉来促进经济的增长。

（3）信息化引发社会生活全面变革。随着全球信息化进程的不断推进，信息技术和信息经济正逐渐成为经济增长和社会进步的主要力量。信息化是一项复杂庞大的系统工程，既涉及信息技术和信息资源本身，也涉及政治体制、经济模式、生活方式、文化传统、人的思维方式和行为等内容。在这个过程中，人类社会生活的方方面面都发生了深刻的变化。托夫勒指出，信息化将是推动社会进步与发展的“第三次浪潮”。

（4）信息化对国际关系产生了深刻影响。信息化的发展不仅促进了国与国之间的联系，同时也改变了国家力量对比并加速多极化格局的形成。

3.1.6　信息化对组织的意义

社会学认为，最有代表性的 4 种组织类型即政府、企业、社团和家庭。其中，信息化对家庭的意义比较单纯，主要是信息消费的方式、内容、价值等的变化。为了讨论问题的方便，我们在这里把家庭排除在外，以下只要提到组织，就是指政府、企业或社团，

而不包括家庭。

信息化对于组织的意义有以下几个方面：

（1）组织的结构创新。一个组织的结构如何进行运作往往是由信息的获取、处理、存储和传递的方式、手段和效率决定的。由于信息化引发的组织创新到处可见，比如，虚拟企业、虚拟社区等。

（2）组织的管理创新。企业应用（Enterprise Resource Planning，ERP）、（Customer Relationship Management，CRM）等信息化管理软件，政府实施电子政务、建立电子政府，社会团体建立电子社区等，都能大大提升管理水平。

（3）组织的经营创新。在信息化的环境下，比较易于做到使经营和管理融为一体，从而提高组织的核心竞争力。

（4）造就信息化的人才队伍。一个组织要实现信息化，就必须首先实现人才信息化。人才信息化有几层含义：一是要造就一支信息化的人才队伍，包括有足够多的精通计算机技术、网络技术和通信技术的专业人才和操作人才；二是要有一批通晓本职业务，并能熟练进行信息系统操作的业务人才；三是要有足够多的同时精通信息技术和经营管理专业的复合型、专家型人才。

3.1.7 信息化的需求

组织对信息化的需求是组织信息化的原动力，它决定了组织信息化的价值取向和成果效益水平。而需求本身又是极为复杂的，它是一个系统性的、多层次的目标体系。

1．组织信息化需求的层次性

一般说来，信息化需求包含三个层次，即战略需求、运作需求和技术需求。

（1）战略需求。组织信息化的目标是提升组织的竞争能力，为组织的可持续发展提供一个支持环境。从某种意义上来说，信息化对组织不仅仅是服务的手段和实现现有战略的辅助工具；信息化可以把组织战略提升到一个新的水平，为组织带来新的发展契机。特别是对于企业，信息化战略是企业竞争的基础。

（2）运作需求。组织信息化的运作需求是组织信息化需求非常重要且关键的一环，它包含三方面的内容：一是实现信息化战略目标的需要；二是运作策略的需要。三是人才培养的需要。

（3）技术需求。由于系统开发时间过长等问题在信息技术层面上对系统的完善、升级、集成和整合提出了需求。也有的组织，原来基本上没有大型的信息系统项目，有的也只是一些单机应用，这样的组织的信息化需求，一般是从头开发新的系统。

组织的三个层次的需求并不是相互孤立的，而是有着内在的联系。信息化需求的获取是一个自上而下的过程，需要对这些需求进行综合分析，才能把握组织对信息化建设的方向。

2．组织信息化需求的系统性

一个组织就是一个复杂的系统。组织的各层次的信息化需求之间存在着有机的内在联系。搞清不同层次需求之间的关系对于组织信息化的实施非常重要，其实，它就是信息化所要解决的问题。各层次信息化需求之间的逻辑关系包括的因果关系、依赖关系、主辅关系和协同关系等。

实现组织信息化是需要资源的，包括人力、物力和财力，以及时间和精力等资源，而任何一个组织所拥有的资源总是有限的，不可能满足所有的需求。在这种情况下，一个组织的信息化应该遵循“总体规划，分步实施”的原则，在多方面、多层次的需求中，首先考虑那些关键的、主要的，并且资源条件允许的需求。另一方面，在组织信息化基础比较薄弱，员工对信息化的认识和技术水平较低的情况下，如果能从相对比较容易实施和产生效果的环节切入，使组织能在短时间内实实在在地体会到信息化所带来的效果，这对组织信息化的推进是很有好处的。我国许多企业在信息化的过程中，首先从工资管理、会计电算化起步，进而开发和应用较为复杂的系统取得了成功，就是一个很好的证明。

3.1.8　信息化战略

中共中央办公厅、国务院办公厅近日印发《2006—2020 年国家信息化发展战略》，提出了到 2020 年我国信息化发展的战略目标。

《战略》提出，到 2020 年，我国信息化发展的战略目标是：综合信息基础设施基本普及，信息技术自主创新能力显著增强，信息产业结构全面优化，国家信息安全保障水平大幅提高，国民经济和社会信息化取得明显成效，新型工业化发展模式初步确立，国家信息化发展的制度环境和政策体系基本完善，国民信息技术应用能力显著提高，为迈向信息社会奠定坚实基础。

《战略》提出了我国信息化发展的九大战略重点。

① 推进国民经济信息化。

② 推行电子政务。

③ 建设先进网络文化。

④ 推进社会信息化。

⑤ 完善综合信息基础设施。

⑥ 加强信息资源的开发利用。

⑦ 提高信息产业竞争力。

⑧ 建设国家信息安全保障体系。

⑨ 提高国民信息技术应用能力，造就信息化人才队伍。

3.2 信息系统工程总体规划

系统规划指根据组织的战略目标和用户提出的需求，从用户的现状出发，经过调查，对所要开发管理信息系统的技术方案、实施过程、阶段划分、开发组织和开发队伍、投资规模、资金来源及工作进度，用系统的、科学的、发展的观点进行全面规划。

3.2.1 信息系统工程总体规划的目标范围

在进行系统规划时，一般应对现行系统进行以下工作。

（1）创造性分析（creative analysis）：对现存问题采用新的方法进行调查分析。

（2）批判性分析（critical analysis）：毫无偏见地仔细询问系统中各组成部分是否有效益或效率，是否应建立新的关系，是否已超越手工作业系统的自动化；询问用户的陈述和假设，选择合理的解决方法；查清及分析有冲突的目标和发展方向。

其目标是从整体上把握管理信息系统的开发，有利于集中全部资源优势，使其得到合理配置与使用；使开发的目标系统与用户建立良好的关系；促进管理信息系统的开发与深化；作为系统开发的标准；促使管理人员回顾过去的工作，发现可以改进的薄弱环节。

信息系统工程总体规划的内容包括：组织的战略目标、政策和约束、计划和指标分析；新的管理信息系统的目标、约束、计划和指标分析、功能结构、组织运行和管理、效益分析和规划；组织的外部环境与管理现状调查；用户的需求调查与分析；新的管理信息系统的描述；新的管理信息系统的运行环境；新的管理信息系统的资源选型；新的管理信息系统的开发计划。

3.2.2 信息系统工程总体规划的方法论

用于管理信息系统规划的方法很多，主要是关键成功因素法（Critical Success Factors，CSF）、战略目标集转化法（Strategy Set Transformation，SST）和企业系统规划法（Business System Planning，BSP）。其他还有企业信息分析与集成技术、产出 / 方法分析、投资回收法、征费法（chargout）、零线预算法和阶石法等。用得最多的是前面三种。

1．关键成功因素法

在现行系统中，总存在着多个变量影响系统目标的实现，其中若干个因素是关键的和主要的（即关键成功因素）。通过对关键成功因素的识别，找出实现目标所需的关键信息集合，从而确定系统开发的优先次序。

关键成功因素来自于组织的目标，通过组织的目标分解和关键成功因素识别、性能指标识别，一直到产生数据字典。

识别关键成功因素，就是要识别联系于组织目标的主要数据类型及其关系。不同的

组织的关键成功因素不同，不同时期关键成功因素也不相同。当在一个时期内的关键成功因素解决后，新的识别关键成功因素又开始。

关键成功因素法能抓住主要矛盾，使目标的识别突出重点。由于经理们比较熟悉这种方法，使用这种方法所确定的目标，因而经理们乐于努力去实现。该方法最有利于确定企业的管理目标。

2．战略目标集转化法

把整个战略目标看成是一个“信息集合”，由使命、目标、战略等组成，管理信息系统的规划过程即是把组织的战略目标转变成为管理信息系统的战略目标的过程。

战略目标集转化法从另一个角度识别管理目标，它反映了各种人的要求，而且给出了按这种要求的分层，然后转化为信息系统目标的结构化方法。它能保证目标比较全面，疏漏较少，但它在突出重点方面不如关键成功因素法。

3．企业系统规划法

信息支持企业运行。通过自上而下地识别系统目标、企业过程和数据，然后对数据进行分析，自下而上地设计信息系统。该管理信息系统支持企业目标的实现，表达所有管理层次的要求，向企业提供一致性信息，对组织机构的变动具有适应性。

企业系统规划法虽然也首先强调目标，但它没有明显的目标导引过程。它通过识别企业“过程”引出了系统目标，企业目标到系统目标的转化是通过企业过程/数据类等矩阵的分析得到的。

3.2.3　信息系统工程总体规划的软件架构组成

在信息系统工程总体规划过程中，软件架构包括好多种形式，下面介绍其中三种：文件服务器、客户/服务器架构、基于 Web 的架构的组成。

1．文件服务器架构

文件服务器架构是一种基于局域网的方案，其中服务器仅仅装载了数据层，系统应用的其他层都在客户端实现。

2．典型的客户/服务器两层架构

在这种架构中，数据和数据处理放在服务器上，而应用领域、表现逻辑和表现层放在客户端。这是真正的两层客户/服务器架构，它充分挖掘使用了客户端的计算能力，并使数据库维护方便，其他客户端可同时使用同一表或数据库的其他记录，大大降低网络流量。但是，应用逻辑必须在所有的客户端进行复制、维护等操作，客户端必须健壮；而且，数据库由众多客户程序直接访问，导致数据库的安全性和完整性难以维护。

3．客户/服务器 N 层架构

客户/服务器 N 层架构就是在客户端和服务器端之间加入一层或者多层应用服务程序（应用服务器）。开发人员把应用的业务逻辑与用户界面分开，将业务逻辑放在经过合理任务划分与物理部署后的中间层应用服务器上，客户程序通过中间层间接地访问数据

库，客户端的修改不影响服务器端；客户程序可以充分扩展；如果需要修改应用程序代码，只需要对中间层应用服务器进行修改，而不用修改客户端应用程序。其好处在于使开发人员更专注于应用系统核心业务逻辑的分析与设计等工作，简化了应用系统的开发、更新等。整个系统架构的可扩展性、数据的安全性等显著增强。三层模型或多层模型可以更好地支持对企业业务逻辑的集中控制与管理。

4．基于 Web 的架构

基于 Web 的架构是松散耦合的，Web 的最大优势在于能够在不同的网络及操作系统中运行，并能方便地扩充到相关企业和最终用户。基于 Web 架构的计算模式本质上借助浏览器和 Web 服务器。它在网络时代将逐步取代传统的软件计算模式，成为目前计算模式的主流。

基于 Web 的架构，把数据表现层逻辑从客户端分离出来部署在 Web 服务器上，应用事务逻辑部署在应用服务器上，数据处理逻辑和数据本身部署在数据服务器上。这种架构，以服务器为中心，客户端瘦小、简单，容易在运行时实现自动升级；应用事务层可在异构的平台的客户端上共享；分离不同的逻辑构件，并采用中间件技术，使得人机交互设计人员、事务逻辑开发人员可以独立地设计和维护他们各自的部分，同时增强了应用系统的动态适应性。

3.2.4 总体规划的实现过程

经过总体规划后，便进入了实现过程阶段。实现过程一般包括以下几个方面：

1．按总体规划报告购置和安装计算机网络系统

购置和安装硬件是件相对简单的事情，只需要按照总体规划报告的要求选择好价格性能比较高的设备，通知供货厂家按需求供货即可。

2．建立数据库系统

如果数据与数据流分析以及数据库设计工作进行得比较规范，而且开发者又对数据库技术比较熟悉的话，按照数据库设计的要求就可以在短时间内搭建一个大型数据库结构。

3．程序设计

目前程序设计的方法大多数采用结构化程序设计方法、原型方法、面向对象的方法进行。

4．系统转换

系统转换，它是指运用某一种方式由新的系统代替旧的系统的过程。因此，在系统转换前，我们必须认真做好各种准备，比如说，系统设备、数据、人员以及有关文件的准备。

5．试运行

系统试运行，它是指在系统没有正式运行之前，选择一些子项目进行的实验运行。

它是系统正式运行的前期准备工作，同时也是系统调试工作的延续。

3.3　信息化的典型应用

3.3.1　政府信息化与电子政务

1．电子政务的概念

电子政务实质上是对现有的、工业时代形成的政府形态的一种改造，即利用信息技术和其他相关技术，将其管理和服务职能进行集成，在网络上实现政府组织结构和工作流程优化重组，超越时间、空间与部门分隔的制约，实现公务、政务、商务、事务的一体化管理与运行。电子政务主要包括三个组成部分：

（1）政府部门内部的电子化和网络化办公。

（2）政府部门之间通过计算机网络进行的信息共享和实时通信。

（3）政府部门通过网络与居民之间进行的双向信息交流。

电子政务的发展过程实质上是对原有的政府形态进行信息化改造的过程，通过不断地摸索和实践，最终构造出一个与信息时代相适应的政府形态。

2．电子政务的内容

在社会中，与电子政务相关的行为主体主要有三个，即政府、企（事）业单位及居民。因此，政府的业务活动也主要围绕着这三个行为主体展开。政府与政府，政府与企（事）业，以及政府与居民之间的互动构成了下面 5 个不同的、却又相互关联的领域。

1）政府与政府（G2G）

政府与政府之间的互动包括首脑机关与中央和地方政府组成部门之间的互动；中央政府与各级地方政府之间；政府的各个部门之间、政府与公务员和其他政府工作人员之间的互动。这个领域涉及的主要是政府内部的政务活动，包括国家和地方基础信息的采集、处理和利用，如人口信息；政府之间各种业务流所需要采集和处理的信息，如计划管理；政府之间的通信系统，如网络系统；政府内部的各种管理信息系统，如财务管理；以及各级政府的决策支持系统和执行信息系统，等等。

2）政府对企（事）业（G2B）

政府面向企业的活动主要包括政府向企（事）业单位发布的各种方针、政策、法规、行政规定，即企（事）业单位从事合法业务活动的环境；政府向企（事）业单位颁发的各种营业执照、许可证、合格证和质量认证等。

3）政府对居民（G2C）

政府对居民的活动实际上是政府面向居民所提供的服务。政府对居民的服务首先是信息服务，让居民知道政府的规定是什么，办事程序是什么，主管部门在哪里，以及各种关于社区公安和水、火、天灾等与公共安全有关的信息。户口、各种证件和牌照的管

理等政府面向居民提供的各种服务。政府对居民提供的服务还包括各公共部门，如学校、医院、图书馆和公园等。

4）企业对政府（B2G）

企业面向政府的活动包括企业应向政府缴纳的各种税款，按政府要求应该填报的各种统计信息和报表，参加政府各项工程的竞、投标，向政府供应各种商品和服务，以及就政府如何创造良好的投资和经营环境，如何帮助企业发展等提出企业的意见和希望，反映企业在经营活动中遇到的困难，提出可供政府采纳的建议，向政府申请可能提供的援助等等。

5）居民对政府（C2G）

居民对政府的活动除了包括个人应向政府缴纳的各种税款和费用，按政府要求应该填报的各种信息和表格，以及缴纳各种罚款等外，更重要的是开辟居民参政、议政的渠道，使政府的各项工作不断得以改进和完善。政府需要利用这个渠道来了解民意，征求群众意见，以便更好地为人民服务。此外，报警服务（盗贼、医疗、急救、火警等）即在紧急情况下居民需要向政府报告并要求政府提供的服务，也属于这个范围。

当前，世界各国电子政务的发展就是围绕着上述 5 个方面展开的，其目标除了不断地改善政府、企业与居民三个行为主体之间的互动，使其更有效、更友好、更精简、更透明之外，更强调在电子政务的发展过程中对原有的政府结构以及政府业务活动组织的方式和方法等进行重要的、根本的改造，从而最终构造出一个信息时代的政府形态。

3．电子政务的技术形式

电子政务在世界范围内的迅速发展经过了近 50 年的信息化进程，西方发达国家政府内部的管理信息系统和各种决策支持系统已经基本完成。当前，电子政务在世界范围内的发展有两个主要的特征：第一个特征是以互联网为基础设施，构造和发展电子政务。第二个特征是，就电子政务的内涵而言，更强调政府服务功能的发挥和完善，包括政府对企业、对居民的服务以及政府各部门之间的相互服务。

电子政务的发展大致经历了以下 4 个阶段。

（1）起步阶段：政府信息网上发布是电子政务发展起步阶段较为普遍的一种形式。大体上是通过网站发布与政府有关的各种静态信息，如法规、指南、手册、政府机构、组织、官员和通信联络等。

（2）政府与用户单向互动：在这个阶段，政府除了在网上发布与政府服务项目有关的动态信息之外，还向用户提供某种形式的服务。

（3）政府与用户双向互动：在这个发展阶段，政府与用户可以在网上完成双向的互动。一个典型的例子是用户可以在网上取得报税表，在网上填完报税表，然后，从网上将报税表发送至国税局。

（4）网上事务处理：沿用上面举过的例子，如果国税局在网上收到企业或居民的报税表并审阅后，向报税人寄回退税支票；或者，在网上完成划账，将企业或居民的退税

所得直接汇入企业或居民的账户。这样，居民或企业在网上就完成了整个报税过程的事务处理。到了这一步，可以说，电子政务在居民报税方面是趋于成熟了。因为，它是以电子的方式实实在在地完成了一项政府业务的处理。

一般来说，电子政务所要处理的业务流有数百个之多。在电子政务的发展中，这数百个业务流的信息化不可能同时进行，更不可能同时趋于成熟；相反地，只能按照轻重缓急，根据需要和可能，一批一批地开发。因此, 建设一个成熟的电子政务可能需要十几年甚至数十年的时间，是一个持续的发展过程。

4．电子政务的应用领域

按照电子政务的应用结构，我国电子政务的应用领域可以集中在以下 6 个方面。

（1）面向社会的应用。主要包括：政府通过自己的网站向社会发布信息，为社会公众提供查询服务；面向社会的各类信访、建议、反馈以及数据收集和统计系统；各类公共服务性业务的信息发布和实施，如工商管理、税务管理、保险管理、城建管理等；面向社会的各类项目的申报、申请；相关文件、法规的发布。

（2）政府部门之间的应用。主要包括：各级政府间的公文信息审核、传递系统；各级政府间的多媒体信息应用平台，如视频会议、多媒体数据交换等；同级政府间的公文传递、信息交换。

（3）政府部门内部的各类应用系统。主要包括：政府内部的公文流转、审核、处理系统；政府内部的各类专项业务管理系统，如日程安排、会议管理、机关事务管理等；政府内部面向不同管理层的统计、分析系统。

（4）涉及政府部门内部的各类核心数据的应用系统。主要包括：机要、秘密文件及相关管理系统；领导事务管理系统，如日程安排等；涉及重大事件的决策分析、决策处理系统；涉及国家重大事务的数据分析、处理系统。

（5）政府电子化采购。也就是政府的电子商务。

（6）电子社区。即城市社区管理中信息手段的应用。

3.3.2　企业信息化与电子商务

1．企业信息化的概念

企业作为国民经济的基本单元，其信息化程度是国家信息化建设的基础和关键。企业信息化就是企业利用现代信息技术，通过信息资源的深入开发和广泛利用，实现企业生产过程的自动化、管理方式的网络化、决策支持的智能化和商务运营的电子化，不断提高生产、经营、管理、决策的效率和水平，进而提高企业经济效益和企业竞争力的过程。

如果从动态的角度来看，企业信息化就是企业应用信息技术及产品的过程，或者更确切地说，企业信息化是信息技术由局部到全局，由战术层次到战略层次向企业全面渗透，运用于流程管理、支持企业经营管理的过程。这个过程表明，信息技术在企业的应

用，在空间上是一个从无到有、由点到面的过程；在时间上具有阶段性和渐进性，起初是战术阶段，经过逐步深化，发展到战略阶段；信息化的核心和本质是企业运用信息技术，进行知识的挖掘和编码，对业务流程进行管理。企业信息化的实施，一般来说，可以沿两个方向进行，一是自上而下，必须与企业的制度创新、组织创新和管理创新结合；二是自下而上，必须以作为企业主体的业务人员的直接受益和使用水平逐步提高为基础。

2．企业信息化的目的

企业信息化的具体目标是优化企业业务活动使之更加有效，它的根本目的在于提高企业竞争能力，使得企业具有平稳和有效的运作能力，对紧急情况和机会做出快速反应，为企业内外部用户提供有价值的信息。企业信息化涉及到对企业管理理念的创新，管理流程的优化，管理团队的重组和管理手段的革新。

（1）技术创新。现实的情况是：一方面，我国企业能够拥有并掌握的技术创新成果甚少，相关信息闭塞。另一方面，又有大量的技术开发成果被沉淀和搁置，造成惊人的浪费。对此，必须运用信息技术，通过在生产工艺设计、产品设计中计算机辅助设计系统的应用，通过互联网及时了解和掌握创新的技术信息，才能加快技术向生产的转化。还有，生产技术与信息技术相结合，能够大幅度地提高技术水平和产品的竞争力。

（2）管理创新。按照市场发展的要求，要对企业现有的管理流程重新整合，从作为管理核心的财务、资金管理，转向技术、物资、人力资源的管理，并延伸到企业技术创新、工艺设计、产品设计、生产制造过程的管理，进而还要扩展到客户关系管理、供应链的管理乃至发展到电子商务。实现这样的管理目标，就必须借助信息技术，发挥计算机的信息采集、储存功能和网络的传递与共享功能。

（3）制度创新。在建立现代企业制度的过程中，信息化起着重要的作用。特别是在由计划经济体制向市场经济体制转轨的过程中，赋予企业信息化一系列特殊的使命，那些不适应企业信息化的管理体制、管理机制和管理制度必须得到创新。同时，通过计算机网络系统管理，建立起明确的岗位责任和精准的监管体系；借助互联网获取全面、系统、及时的信息，彻底改变企业一直沿用的计划经济体制的资源分配方式和管理方式，注重市场信息的分析和研究，提供准确及时的决策信息；应用科学的方法实施管理。因此，建立在计算机网络技术基础上的管理，才更科学、更有效。我们在倡导企业技术改造、技术创新的同时，还应当倡导企业加快管理改造和管理创新。

3．企业信息化的规划

企业信息化一定要建立在企业战略规划基础之上，以企业战略规划为基础建立的企业管理模式是建立企业战略数据模型的依据。

企业信息化就是技术和业务的融合。这个“融合”并不是简单地利用信息系统对手工的作业流程进行自动化改造，而是需要从三个层面来实现。

（1）企业战略的层面。在规划中必须对企业目前的业务策略和未来的发展方向作深入分析。通过分析，确定企业的战略对企业内外部供应链和相应管理模式，从中找出实

现战略目标的关键要素，分析这些要素与信息技术之间的潜在关系，从而确定信息技术应用的驱动因素，达到战略上的融合。

（2）业务运作层面。针对企业所确定的业务战略，通过分析获得实现这些目标的关键驱动力和实现这些目标的关键流程。这些关键流程的分析和确定要根据它们对企业价值产生过程中的贡献程度来确定。关键的业务需求是从那些关键的业务流程分析中获得的，它们将决定未来系统的主要功能。这一环节非常重要，因为，信息系统如果能够与这些直接创造价值的关键业务流程相融合，这对信息化投资回报的贡献是非常巨大的，也是信息化建设的成败的一个衡量指标。

（3）管理运作层面。虽然这一层面从价值链的角度上来说，是属于辅助流程，但它对企业日常管理的科学性、高效性是非常重要的。另外，在企业战略层面的分析中，我们可以获得适应企业未来业务发展的管理模式，这个模式的实现是离不开信息技术的支撑的。所以，在管理运作层面的规划上，除了提出应用功能的需求外，还必须给出相应的信息技术体系，这些将确保管理模式和组织架构适应信息化的需要。

企业信息化规划的重要性是不言而喻的，但是，要防止另一种倾向，就是把信息化规划片面地理解为信息技术规划。

企业战略数据模型分为数据库模型和数据仓库模型，数据库模型用来描述日常事务处理中数据及其关系；数据仓库模型则描述企业高层管理决策者所需信息及其关系。在企业信息化过程中，数据库模型是基础，一个好的数据库模型应该客观地反映企业生产经营的内在联系。数据库是办公自动化、计算机辅助管理系统、开发与设计自动化、生产过程自动化、Intranet 的基础和环境。

信息技术和网络技术都在飞速发展，企业信息化是多种类、多层次信息系统建设、集成和应用的过程，因而，不是一蹴而就的事情，需要结合企业的实际，全面规划，分步实施。

4．企业信息化方法

通过二三十年的发展，人们已经总结出了许多非常实用的企业信息化方法，并且还在探索新的方法。这里只简单介绍几种常用的企业信息化方法。

1）业务流程重构方法

企业业务流程重构的中心思想是，在信息技术和网络技术迅猛发展的时代，企业必须重新审视企业的生产经营过程，利用信息技术和网络技术，对企业的组织结构和工作方法进行“彻底的、根本性的”重新设计，以适应当今市场发展和信息社会的需求。

2）核心业务应用方法

任何一家企业，要想在市场竞争的环境中生存发展，都必须有自己的核心业务，否则，必然会被市场所淘汰。当然，不同的企业，其核心业务是不同的。比如，一个石油生产企业，原油的勘探开发生产就是它的核心业务。围绕核心业务应用计算机技术和网络技术是很多企业信息化成功的秘诀。

3）信息系统建设方法

对大多数企业来说，建设信息系统是企业信息化的重点和关键。因此，信息系统建设成了最具普遍意义的企业信息化方法。

4）主题数据库方法

主题数据库就是面向企业业务主题的数据库，也就是面向企业的核心业务的数据库。有些企业，特别是在业务数量浩繁，流程错综复杂的大型企业里，建设覆盖整个企业的信息系统往往很难成功，但是，各个部门的局部开发和应用又有很大弊端，会造成系统分割严重，形成许多“信息孤岛”，造成大量的无效或低效投资。在这样的企业里，应用主题数据库方法推进企业信息化无疑是一个投入少、效益好的方法。

5）资源管理方法

计算机技术和网络技术的应用为企业资源管理提供了强大的能力。目前，流行的企业信息化的资源管理方法有很多，最常见的有 ERP（Enterprise Resource Planning，企业资源计划）、SCM（Supply Chain Management，供应链管理）等。

6）人力资本投资方法

人力资本的概念是经济学理论发展的产物。人力资本与人力资源的主要区别是人力资本理论把一部分企业的优秀员工看做是一种资本，能够取得投资收益。

人力资本投资方法特别适用于那些依靠智力和知识而生存的企业，例如，各种咨询服务、软件开发等企业。

3.3.3 企业资源规划的结构和功能

在制造业中，物料需求计划（Material Requirement Planning，MRP）在 20 世纪 70 年代中期成为了生产管理和控制中的基本概念。在这个阶段物料单系统（Bill of Materials，BOM）是主流，它主要包括通过利用目录来实施对订单的管理。MRP 的概念逐渐发展，从物料订货、库存管理到工厂和人力资源计划以及分销计划，然后再进一步发展成为制造资源计划（Manufacturing Resource Planning II，MRPII）。在此基础上加入财务会计功能和人力资源管理功能、销售功能和管理功能等，就成为在全球各种行业企业的信息系统中的主流，也就是企业资源计划。

1．ERP 的概念

企业的所有资源包括三大流：物流、资金流和信息流。ERP 也就是对这三种资源进行全面集成管理的管理信息系统。概括地说，ERP 是建立在信息技术基础上，利用现代企业的先进管理思想，全面地集成了企业的所有资源信息，并为企业提供决策、计划、控制与经营业绩评估的全方位和系统化的管理平台。ERP 系统是一种管理理论和管理思想，不仅仅是信息系统。它利用企业的所有资源，包括内部资源与外部市场资源，为企业制造产品或提供服务创造最优的解决方案，最终达到企业的经营目标。

ERP 理论与系统是从 MRP-II 发展而来的。MRP-II 的核心是物流，主线是计划，但

ERP 已将管理的重心转移到财务上，在企业整个经营运作过程中贯穿了财务成本控制的概念。ERP 极大地扩展了业务管理的范围及深度，包括质量、设备、分销、运输、多工厂管理、数据采集接口等。ERP 的管理范围涉及企业的所有供需过程，是对供应链的全面管理。企业运作的供需链结构，如图 3-1 所示。

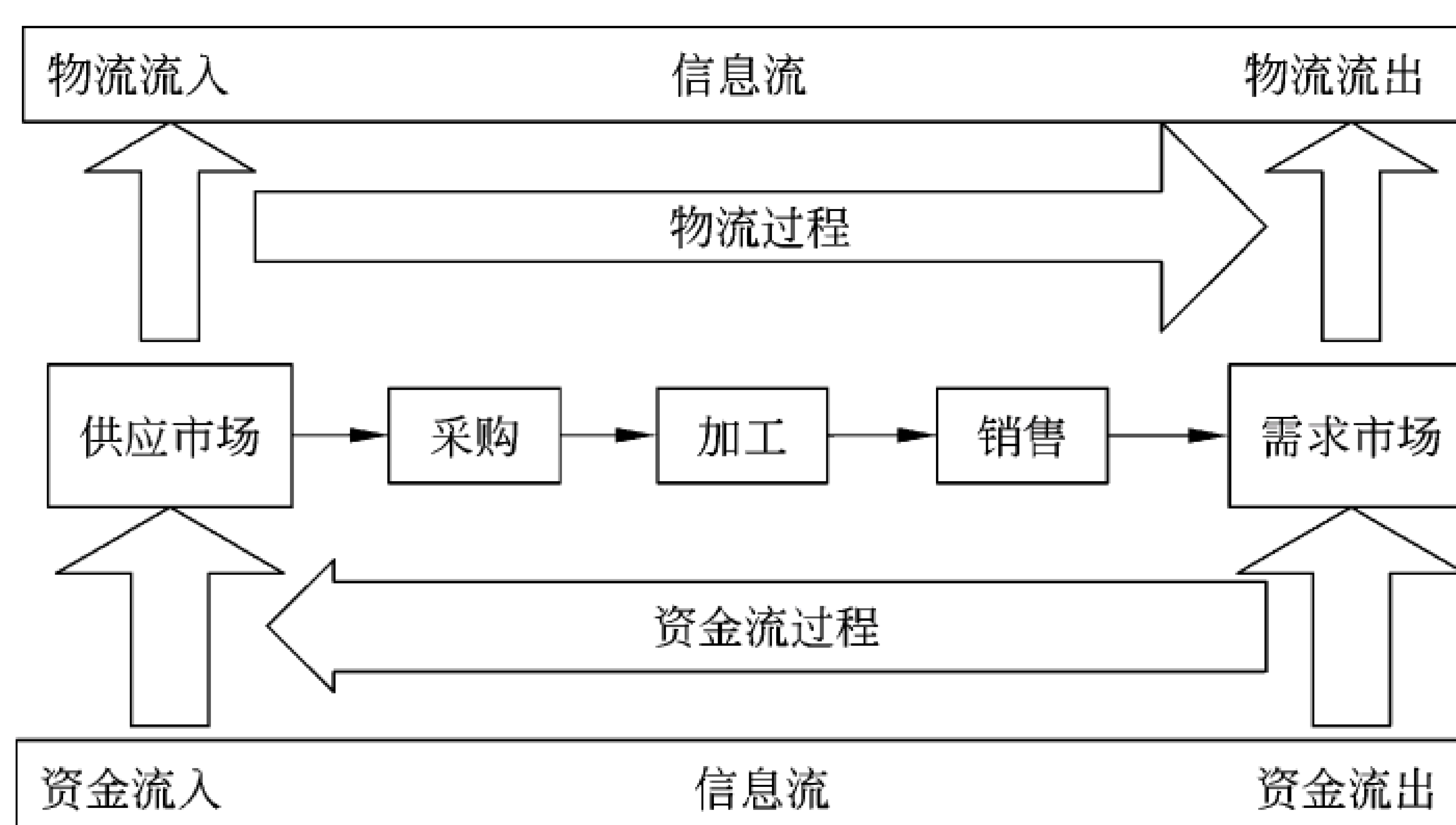


图 3-1　企业运作的供需链图

2．ERP 的结构

ERP 中的企业资源包括企业的“三流”资源，即物流资源、资金流资源和信息流资源。ERP 实际上就是对这“三流”资源进行全面集成管理的管理信息系统。

ERP 的结构原理如图 3-2 所示 。由图可知，ERP 主要包括了以下模块。

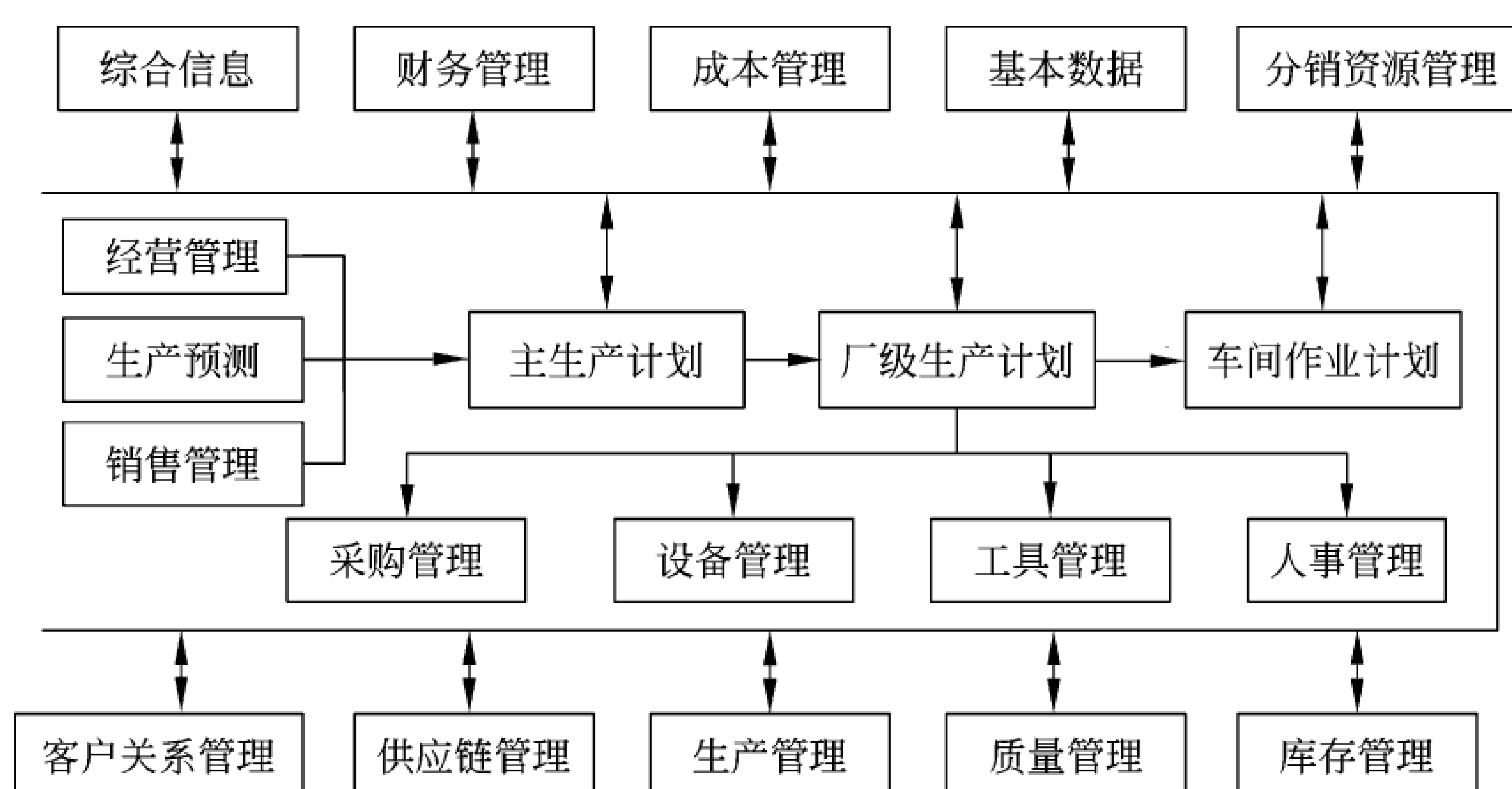


图 3-2　ERP 结构原理图

1）生产预测

市场需求是企业生存的基础，在 ERP 中首先需要对市场进行较准确的预测。预测主要用于计划，在 ERP 的 5 个层次的计划中，前三个层次计划，即经营计划、生产计划大

纲和主生产计划的编制都离不开预测。常用的预测方法有德尔菲（Delphi）方法、移动平移法、指数平滑法和非线性最小二乘曲线拟合法。

2）销售管理（计划）

销售管理主要是针对企业的销售部门的相关业务进行管理。企业销售部门是企业与市场连接的桥梁，其主要职能是为客户和最终用户提供服务，从而使企业获得利润，实现其经济和社会价值。销售管理从其计划角度来看，属于最高层计划的范畴，是企业最重要的决策层计划之一。

3）经营计划（生产计划大纲）

生产计划大纲（Production Planning，PP）是根据经营计划的生产目标制定的，是对企业经营计划的细化，用以描述企业在可用资源的条件下，在一定时期中的产量计划。生产计划大纲在企业决策层的三个计划中有承上启下的作用，一方面它是企业经营计划和战略规划的细化，另一方面它又用于指导企业编制主生产计划，指导企业有计划地进行生产。

4）主生产计划

主生产计划（Master Production Schedule，MPS）是对企业生产计划大纲的细化，说明在一定时期内的如下计划：生产什么，生产多少和什么时候交货。主生产计划的编制以生产大纲为准，其汇总结果应当等同于生产计划大纲，同时，主生产计划又是其下一层计划——物料需求计划的编制依据。

主生产计划的编制是 ERP 的主要工作内容。主生产计划的质量将大大影响企业的生产组织工作和资源的利用。

5）物料需求计划

物料需求计划是对主生产计划的各个项目所需的全部制造件和全部采购件的网络支持计划和时间进度计划。它根据主生产计划对最终产品的需求数量和交货期，推导出构成产品的零部件及材料的需求数量和需求时期，再导出自制零部件的制作订单下达日期和采购件的采购订单发送日期，并进行需求资源和可用能力之间的进一步平衡。物料需求计划是生产管理的核心，它将主生产计划安排生产的产品分解成各自制零部件的生产计划和采购件的采购计划。物料需求计划属于 ERP 管理层计划。

6）能力需求计划

能力需求计划（Capacity Requirements Planning，CRP）是对物料需求计划所需能力进行核算的一种计划管理方法。旨在通过分析比较 MRP 的需求和企业现有生产能力，及早发现能力的瓶颈所在，为实现企业的生产任务而提供能力方面的保障。

7）车间作业计划

车间作业计划（Production Activity Control，PAC）是在 MRP 所产生的加工制造订单（即自制零部件生产计划）的基础上，按照交货期的前后和生产优先级选择原则以及车间的生产资源情况（如设备、人员、物料的可用性、加工能力的大小等），将零部件的

生产计划以订单的形式下达给适当的车间。车间作业计划属于 ERP 执行层计划。当前主流的车间作业计划模式是 JIT（Just In Time）模式。

8）采购与库存管理

采购与库存管理是 ERP 的基本模块，其中采购管理模块是对采购工作——从采购订单产生至货物收到的全过程进行组织、实施与控制，库存管理（Inventory Management，IM）模块则是对企业物料的进、出、存进行管理。

9）质量与设备管理

质量管理贯穿于企业管理的始终。企业经营活动中的各环节、各项工作以及各种产品都离不开质量，都要讲究质量。全面质量管理（Total Quality Management，TQM）是质量管理的主要实施模式，它要求对企业的全过程进行质量管理，而且明确指出执行质量职能是企业全体人员的责任。

设备管理是指依据企业的生产经营目标，通过一系列的技术、经济和组织措施，对设备寿命周期内的所有设备物资运动形态和价值运动形态进行的综合管理。

10）财务管理

会计工作是企业管理的重要组成部分，是以货币的形式反映和监督企业的日常经济活动，并对这些经济业务的数据进行分类、汇总，以便为企业管理和决策提供必要的信息支持。企业财务管理是企业会计工作和活动的统称，财务管理是一种综合性的管理，它渗透在企业全面的经济活动之中，哪里有经济活动，哪里就有资金运动，哪里就有财务管理。

11）ERP 有关扩展应用模块

如客户关系管理、分销资源管理、供应链管理和电子商务等。这几个扩展模块本身也是一个独立的系统，在市场上它们常作为独立的软件产品进行出售和实施。

3．ERP 的功能

ERP 为企业提供的功能是多层面的和全方位的。

（1）支持决策的功能。ERP 在 MRP-II 的基础上扩展了管理范围，给出了新的结构，将企业内部业务流程划分成几个相互协同作业的支持子系统，如财务、市场营销、生产制造、质量控制、服务维护和工程技术等，并在功能上增加了质量控制、运输、分销、售后服务与维护，以及市场开发、人事管理等功能，把企业的制造系统、营销系统、财务系统等都紧密地结合在一起，可以实现全球范围内的多工厂、多地点的跨国经营运作，因而，能够不断地收到来自各个业务过程的运作信息，并且提供了对质量控制、市场变化、客户满意度、经营效绩等关键问题的实时分析，从而有力地支持企业的各个层面上的决策。

（2）为处于不同行业的企业提供有针对性的 IT 解决方案。ERP 打破了 MRP-II 只局限在传统制造业的格局，把应用扩展到其他行业，并逐渐形成了针对于某种行业的解决方案。有些 ERP 供应商除了传统的制造业解决方案外，还推出了商业与零售业、金融业、

能源、公共事业、工程与建筑业等行业的解决方案，以财务、人事、后勤等功能为核心，加入每一行业特殊的需求。

（3）从企业内部的供应链发展为全行业和跨行业的供应链。当前企业只有联合该行业中其他上下游企业，建立一条业务关系紧密、经济利益相连的供应链实现优势互补，才能适应社会化大生产的竞争环境，共同增强市场竞争实力。因此，供应链的概念就由狭义的企业内部业务流程扩展为广义的全行业供应链及跨行业的供应链，ERP 的管理范围亦相应地由企业的内部拓展到整个行业的原材料供应、生产加工、配送环节、流通环节以及最终消费者。在整个行业中建立一个环环相扣的供应链，使多个企业能在一个整体的 ERP 管理下实现协作经营和协调运作。把这些企业的分散计划纳入整个供应链的计划中，从而大大增强该供应链在大市场环境中的整体优势，同时也使每家企业之间均可实现以最小的个别成本和转换成本来获得成本优势。

3.3.4 客户关系管理在企业的应用

1．CRM 的概念

客户关系管理（Customer Relationship Management，CRM） 是涵盖构建良好客户关系所应具备所有要素的一门科学。从管理科学的角度考察，CRM 源于市场营销理论，从解决方案的角度考察，CRM 是将市场营销的科学管理理念通过信息技术的手段集成在软件上面，得以在全球大规模的普及和应用。

CRM 主要包含以下 4 个内容：提供的信息要有利于更好地理解客户；流程管理要为客户提供高效、适当的体验；允许员工使用以上知识的软件；培训并改变管理要素，使员工和企业了解并且有能力提供那些构建强有力关系、提高客户忠诚度的体验。

CRM 的目的是提高收入。CRM 通过管理客户与企业之间的关系（包括营销、销售、服务和维护）使企业达到并超过客户期望来提高客户忠诚度进而提高收入。客户关系管理不仅仅是使客户满意，理解客户或流程自动化，它致力于将 4 种核心要素（信息、流程、技术和人员）相结合，提供一组持续而积极的可控个人化体验来提高客户忠诚度。

CRM 的核心思想就是以客户为中心。CRM 的宗旨就是改善企业与客户之间的关系，使客户时时刻刻感觉到企业的存在，企业随时了解客户的变化。CRM 要求企业从传统的“以产品为中心”的经营理念解放出来，确立“以客户为中心”的企业运作模式，从而提高客户的忠诚度，为企业带来丰厚的利润和上升空间。可见，CRM 就是指企业通过富有意义的交流沟通，理解并影响客户行为，最终实现客户保留、客户忠诚和客户创利的目的，是一个将客户信息转化为积极的客户关系的反复循环的过程。

2．CRM 的背景

CRM 的产生，是市场需求和管理理念更新的需要，是企业管理模式和企业核心竞争力提升的要求，使电子化浪潮和信息技术支持等因素推动和促成的结果。

1）管理理念的更新

客户关系管理指的是以客户为中心，及时地提供产品和服务，提高客户的满意程度，最大限度的减少客户流失，保持较高的市场竞争能力和盈利能力，实现客户和企业双方获利的一种管理方法。客户满意能够形成长期的合作关系，能够实现客户和企业的“双赢”。客户关系管理实质上是经营理念的升华，也是企业在市场竞争加剧的条件下，遵循市场发展规律的必然选择。

另外，在互联网时代，信息技术革命的影响已由纯科技领域向市场竞争和企业管理各领域全面转变。这一转变对企业市场营销管理中的传统观念和行为产生了巨大的冲击，也为市场营销管理思想的普及和应用开辟了广泛的前景，并在此基础上产生了大量新的营销管理理念，如数据库营销、关系营销、一对一营销等，将我们带入一个全新的电子商务时代。

2）市场需求的拉动

从产品时代起，产业的长期盈利吸引了新企业的不断进入，以及一个行业内企业及其提供的产品和服务不断增加进一步加剧了市场竞争，原本稀缺的市场供应的产品、服务逐渐变得饱和，而客户资源逐渐相对变得稀少，企业和客户的地位也随之发生相应的改变。此时市场的主动权被让给了客户，企业只有赢得客户才能赢得市场。

因此，企业相应的改变了经营策略。企业无法再像以前那样根据自己所能提供的商品或服务满足客户现实存在的多样化需求，而是在了解市场和客户真实需要的基础上提供令其满意的产品和服务，供需的信息流动变为客户和企业实时交流信息，客户将需求情况传达给企业，企业根据客户的需求信息进行设计、生产和服务，客户能够根据自己的需求量身定做适合自己需要的产品和服务。

由于市场环境的这种变化，企业在其目前的制度体系和业务流程中出现了种种难以解决的问题。比如业务人员无法跟踪众多复杂和销售周期长的客户。这一系列问题的产生，使越来越多的企业要求销售与服务的日常业务自动化和科学化，这是客户关系管理应运而生的需求基础。

3）信息技术的推动

随着信息技术的发展，企业核心竞争力对于企业信息化的程度和管理水平的依赖越来越高，这就需要企业主动开展组织架构、工作流程的重组，同时对面向客户的各项信息进行集成，组建以客户为中心的企业，实现对客户的全面管理。客户信息是客户关系管理的基础。近年来，随着数据库技术的应用与数据仓库、商业智能、知识发现等技术的发展，提高了收集、整理、加工和利用客户信息的质量。

电子商务在全球范围内正开展得如火如荼，正在改变企业的经营方式。信息技术和 Internet 成为日渐成熟的商业手段和工具，越来越广泛的应用于金融、证券、电信、电力和商业机构等各个行业领域的信息系统的构建，其应用领域也从传统的办公事务处理发展到在线分析、决策支持、Internet 内容管理和应用开发等。客户关系管理由此被视作电

子商务的主要推断力量，并领导着电子商务的革命，更被视为企业实现电子商务、客户服务和销售自动化的最佳途径。通过先进的管理理念和软件不仅能够彻底改变企业的管理和运营模式，也直接影响到企业竞争力的强弱。

3．CRM 的内容

CRM 是一套先进的管理思想及技术手段，它通过将人力资源、业务流程与专业技术进行有效的整合，最终为企业涉及到客户或消费者的各个领域提供了完美的集成，使得企业可以更低成本、更高效率地满足客户的需求，并与客户建立起基于学习性关系基础上的一对一营销模式，从而让企业可以最大程度提高客户满意度及忠诚度。在此，我们简单的介绍一下 CRM 系统的主要模块。

1）销售自动化

销售自动化（Sales Force Automation，SFA）是 CRM 中最基本的模块。SFA 是早期的针对客户的应用软件的出发点，但今天其范围已经大大的扩展了，它以整体的视野，提供继承性的方法来管理客户关系。

SFA 主要是提高专业销售人员的大部分活动的自动化程度。它包含一系列的功能，提高销售过程的自动化程度，并向销售人员提供工具，提高其工作效率。它的功能一般包括日历和日程安排、联系和客户管理、佣金管理、商业机会和传递渠道管理、销售管理、建议的产生和管理、定价、区域划分、费用报告等。

一个典型的 SFA 系统除了日常管理功能外，也集成了其他信息源，供销售人员随时调用，主要包括产品目录和价格、购买记录、服务记录、存货情况、促销文本资料、信用记录。SFA 应用往往集成如电子邮件、办公软件等其他各种标准应用，使用户可以在同一界面内完成各种工作。它支持各种流行的客户终端，销售人员可以根据需要选择适用的设备。

2）营销自动化

营销自动化（Marketing Automation，MA）模块作为对 SFA 的补充，它为营销提供了独特的能力，如营销活动计划的编制和执行、计划结果的分析。它集成客户商业智能信息、产品信息、“营销百科全书”等信息源，“营销百科全书”是一个提供了全面营销信息的仓库，包括产品、技术特点、各种文本宣传资料以及产品使用手册等信息。营销自动化模块与 SFA 模块的不同在于，它们提供的功能不同，这些功能的目标也不同。营销自动化模块不局限于提高销售人员活动的自动化程度，其目标是为营销及其相关活动的设计、执行和评估提供详细的框架。

3）客户服务与支持

在很多情况下，客户的保持和提高客户利润贡献度依赖于提供的优质服务。因此客户服务和支持对企业来说是极为重要的。在 CRM 中，客户服务与支持主要是通过呼叫中心（call center）和互联网来实现，在满足客户的个性化要求方面，它们是以高速度、准确性和高效率来完成客户服务人员的各种要求。

CRM 系统中的强有力的客户数据使通过多种渠道（如互联网、呼叫中心）的纵横销售变为可能，当把客户服务与支持功能同销售、营销功能比较好地结合起来时，就能为企业提供很多好机会，向已有的客户销售更多的产品。客户服务与支持的内容应包括：客户关怀；纠纷、订货、订单跟踪；现场服务；问题及其解决方法的数据库；维修行为安排和调度；服务协议和合同；服务请求管理等。

4）商业智能

商业智能是指利用数据挖掘、知识发现等技术分析和挖掘结构化的、面向特定领域的存储与数据仓库的信息，它可以帮助用户认清发展趋势、识别数据模式、获取职能决策支持、得出结论。商业智能的范围包括客户、产品、服务和竞争者等。在 CRM 系统中，商业智能主要是指客户智能（customer intelligence）。利用客户智能，可以收集和分析市场、销售、服务和整个企业的各类信息，对客户进行全方位的了解，从而理顺企业资源与客户需求之间的关系，增强客户的满意度和忠诚度，实现获取新客户、支持交叉销售、保持和挽留老客户、发现重点客户、支持面向特定客户的个性化服务等目标，提高盈利能力。

4．CRM 的解决方案和实施过程

到目前为止，CRM 尚未有成型的理论出现，各大企业在开发 CRM 系统时，各展其能，导致了市场上的 CRM 系统各不相同。但与此同时，各厂商对 CRM 系统认识上的一致性又使他们的解决方案存在一定的共同之处。

目前国内外产品一般都具有如下特点：

（1）通常都支持电子商务的销售方式（这里指的电子商务是以电子流的方式进行销售活动的商业模式，如网上购物）。

（2）CRM 的基本构成通常包括以下 4 部分。

- 销售管理：对销售队伍、销售机会以及销售业务的管理。
- 市场管理：对市场的设定、追踪和分析总结。
- 服务管理：对服务活动的信息支持，包括对日程的安排、服务活动的监控以及知识库。
- 现场服务管理：为游离于企业之外的现场服务人员利用移动设备检索服务的安排以及关于产品、客户等与服务有关的信息。

（3）CRM 的辅助构成。在上述基本框架的基础上一些国外的软件厂商还纳入以下部分。

- 电子商务支持 CRM 系统，不仅提供电子商务接口，还全面开发电子商务。
- 呼叫中心支持由合作的硬件厂商参与并提供全套设备，而不仅仅是提供支持呼叫中心的应用软件。
- 对移动设备的支持。

CRM 系统除了组成部分的要求外，在技术上需要实现其特有的一些功能，Hurwitz

Group 给出了 CRM 的 6 个主要功能和技术要求，如图 3-3 所示。

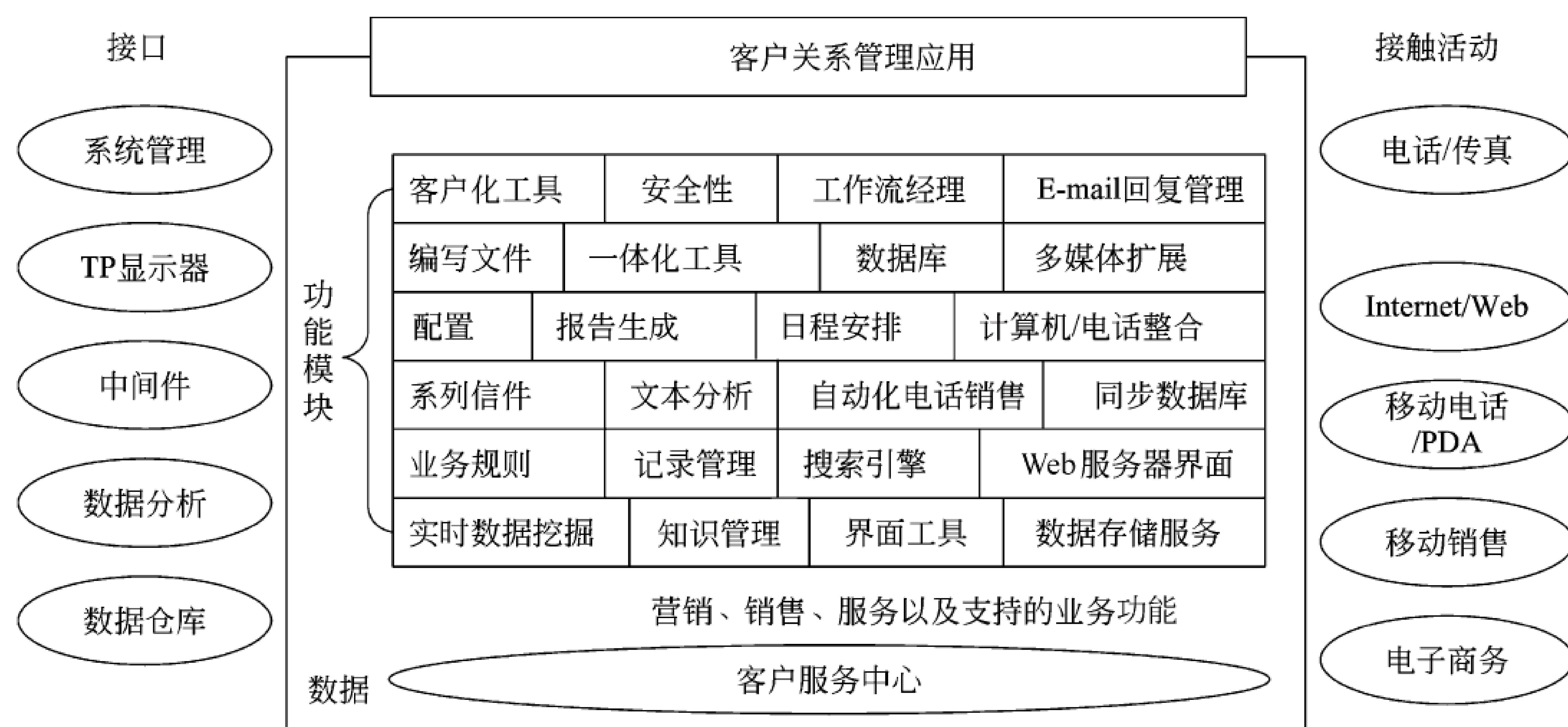


图 3-3 CRM 软件系统的技术功能

（1）信息分析能力，CRM 系统有大量关于客户和潜在客户的信息，企业应该能充分利用这些信息，使得决策者所掌握的信息完全，从而能更及时地做出决策。

（2）对客户互动渠道进行集成的能力，不管客户由何种渠道与企业联系，与客户的互动都应该是无缝的、统一的、高效的。

（3）支持网络应用的能力。

（4）建设集中的客户信息仓库的能力，CRM 解决方案采用集中化的信息库，这样所有与客户接触的雇员都可获得实时的客户信息，而且使得各业务部门和功能模块间的信息能统一起来。

（5）对工作流进行集成的能力，CRM 解决方案应该能具有很强的功能，为跨部门的工作提供支持，使这些工作能够动态地、无缝地完成。

（6）与 ERP 功能的集成，CRM 要与 ERP 在财务、制造、库存、分销、物流和人力资源等环节连接起来，从而提供一个闭环的客户互动循环。

全球各大企业都在实施客户关系管理，实施中有以下 4 个问题很重要：

（1）寻找正确的客户。

（2）提供正确的产品和服务。

（3）在正确的时间与客户接触。

（4）利用正确的渠道为客户提供服务。

成功实施 CRM 的 8 个战略阶段这一战略性方法多年来得到成功运用，是项目成功的关键所在：

（1）分析与规范。实施过程阶段包括确定一个综合性的需求分析，确定项目范围和

系统规范。

（2）项目计划和管理。项目实施计划在这一阶段得以制定。供应商项目管理者应是供应商同企业之间的沟通点。另外，还需任命一名来自企业的系统管理员，作为内部系统专家。这一阶段还包括组建和培训项目工作组。最后必须将投资回报率量化，以有效地衡量新系统所带来的投资收益。

（3）系统配置与修改。在本阶段，CRM 系统将得到配置和修改，以适应具体的商业需要。经特殊调整的系统必须伴随技术培训，使员工能尽量地自己解决技术问题。同时，所有新的软硬件都应在本阶段安装好。另外，对系统进行的所有必要修改都在此阶段完成。

（4）原型、兼容测试和系统重复运行。本阶段包括系统原型的建立和测试。企业员工将在此阶段熟悉安装程序和所安装系统的方方面面。数据转换这一关键任务也属于这一阶段。供应商的实施专家和本公司的 MIS 人员之间将进行大量的沟通。由于数据转换过程工作量极大，因此要精确预测该过程的时间表几乎是不可能的。

（5）主导系统和质量保证测试。这一阶段包括大量的培训，让公司自己的员工来培训员工如何使用 CRM 系统。这位"培训者"应负责培训所有的终端用户和管理层如何使用新系统。"培训者"必须接受由软件供应商进行的培训，成为新系统专家。开始应同小型的用户全体合作，对新系统进行测试。由企业员工参与的对新系统进行测试的质量保险测试应制成文档，提供给项目工作组管理人员，这样系统即可实现平稳过渡。

（6）最后实施和推广。这是一个行动阶段，是新系统的最终实施阶段。这些最后步骤会花去技术人员大量的时间。应准备好一份实施指南，简单列出实施前或实施过程中必须完成的每一项任务。本阶段还包括对所有用户的正规培训。用户必须认识到使用新系统的即时和明显的好处。培训必须以计划阶段确定的需要为基础。一个执行良好的培训计划决定着成败。

（7）持续支持。对系统的持续支持要求公司配备至少一名全职的内部系统管理员，这样便可保证技术上自给自足的灵活性。系统管理员应从计划阶段就开始接触 CRM 系统。因为 CRM 系统的技术支持是艰巨的工作，所以务必让供应商提供综合性的支持计划，对内部工作组也要进一步补充和完善。

（8）系统的持续管理。CRM 系统基础设施一定要提供业绩管理衡量标准。该系统必须有效地获取适当数据，并为接触的每个个体提供途径。为保证系统带来所希望的益处，在将其推广到所有用户之前一定要加以测试。最后，系统还应为监管指导委员会和项目工作组提供反馈。

5．CRM 的价值

CRM 作为一个经营理念，它在企业范围内的实践最终是为了实现企业所制定的经营目标。那么 CRM 的应用在帮助企业实现经营目标的过程中能取得怎样的效益杠杆作用呢？

1）提高内部员工的工作效率，节省日常开支

这是一个最明显的投资回报（Return Over Investment，ROI）指标，节省开支其实等同于利润的增加。

2）提高客户满意度

使用企业的 CRM 系统可以让客户的满意度有所提升，例如企业的各种自助服务让客户可以不受上班时间限制，提高了客户进行各种查询、购买活动的灵活性。

3）提高客户的忠诚度

一般来说，任何技术应用都比不上企业员工对客户真诚的、通情达理的态度，在这一点上，企业通过 CRM 系统的技术应用，可以在不同程度上提高客户对企业的依赖性。

除了上述满意和忠诚会给企业带来间接效益外，CRM 还可以利用 CRM 对客户的分析能力，打好营销、促销战役来获得利润的提高。

3.3.5 企业门户

“企业门户”这一术语被广泛引用，但至今尚不能给其下一个精确的定义，因为业界还没有一个公认的“门户标准”。一般认为企业门户就是一个信息技术平台，这个平台可以提供个性化信息服务，它联接企业的内部和外部，为企业提供一个单一的访问企业各种信息资源和应用程序的入口，企业的员工、客户、合作伙伴和供应商都可以通过这个门户获得个性化的信息和服务。

1．企业门户的功能

门户的主要功能如下。

1）个性化

对于门户功能的一个最强烈要求就是它应具有定做用户工作区的内容和外观的能力，门户的个性化功能包括演示个性化、个性化的信息过滤、个性化的用户配置。

2）演示功能

对信息进行组织并简化信息消费流程是门户的首要目标。显示方式是门户功能的关键部分，舒适且符合人机工程学的显示方式是提高信息消费效率的首要因素。一个高效门户的第二个目标是显示要尽可能的直观，让用户能够易学、易用。

3）知识及内容的创建与管理

公司内部的互联网通常是公司建立的第一个知识库，同时门户应当允许各类用户使用各种各样的工具来创建内容，这样就从原来只有管理者能够创建内容变为以用户为基础进行内容创建和发布。

4）搜寻和检索

门户平台厂商允许通过各种不同的搜索机制和算法来改善原有的信息搜索方法。可以通过提供以下功能来解决这个问题：

- 全文本搜索。

- 允许在全部信息资源中进行搜索，包括文件、数据库、因特网及更多。
- 根据主题进行搜索。
- 对信息进行分类搜索。

5）元数据管理/分类

门户运用各种不同的技术来进行信息及其资源的组织及分类工作，并用元数据定义企业内部的通用语言，通过使用元数据，信息的含义不再模棱两可或混乱不堪。

6）查询/报告和分析

许多起源于商业智能市场的门户提供了自己的信息查询和报告工具，这并非企业门户的最基本的共同特征。

7）数据管理和应用集成

门户产品利用各种工具对已有的应用程序及信息资源进行整合，通过实现信息管理以及信息访问的功能使门户成为访问企业 IT 环境的中心入口。当前，人们期望用统一的视图来整合所有的事件，不管这些事件是结构性数据还是无结构性数据，是过程还是程序。

8）文档管理

文档管理功能是人们广泛需要的另一项功能。

9）协同和信息共享/知识汲取及索引

在企业的商业流程中总是涉及到其成员之间的协同问题。在线讨论和聊天室可以被看作是消费者门户中一个普通的会议室，企业门户也正努力学习和利用这些功能来提高企业的信息共享与协同能力。

10）虚拟社区

在门户中引入社区建设与合作功能可以使存在于企业文档、知识库和其他数据源中的信息保持及时性。使得用户可以迅速的获得和交流及时、相关、有用的信息。

11）流程支持

在短期内，是否支持企业商务流程自动化或许可以成为区分不同门户技术的关键所在。把商务流程信息与工作流技术相融合以支持企业的营运将是门户可以提供的另一个强大功能。

12）商业功能/垂直市场应用程序

跟门户平台最初的起源有关，一些门户为一些特别的商业活动或相关的垂直市场应用程序提供特别的专家系统或功能。

13）集中式目录支持和门户管理

为了加强对目录的管理和安全控制，许多企业采用了简便目录访问协议结构（Lightweight Directory Access Protocol，LDAP）。并不是所有的门户操作系统平台都支持集中式目录支持功能，但对于任何想广泛参与公司的门户部署竞争的厂商来说，他们的门户必须要支持这种功能。

14）安全功能

为了实现提供一种统一的企业信息资源访问入库的承诺，企业门户必须采用有效的安全层来保护这些资源。

2．企业门户的分类

现在有众多的分别从事企业资源计划和商务智能开发以及文件管理和应用程序集成的计算机软件提供商使用“企业门户”来称呼他们的产品，尽管这些产品展现出的特点各不相同，但他们提供的企业门户种类不外乎下面几大类型。

1）企业信息门户

企业信息门户（Enterprise Information Portal，EIP）重点强调的是为访问结构数据和无结构数据提供一个统一入口，它强调对结构化与非结构化数据的收集、访问、管理和无缝集成。企业信息门户的目的是通过一个个性化、集中式的信息浏览手段，使企业员工、合作伙伴、客户、供应商都能够访问企业内部网络和因特网存储的各种自己所需的信息。

在目前企业门户的应用中，信息门户是企业比较认同的。然而，由于企业信息门户侧重于数据本身，所以其支持企业的商业流程或资料整理的能力受到了限制。这类企业门户广泛使用的名称还有公司门户（Corporate Portal）和商业门户（Business Portal）。

2）企业知识门户

企业知识门户（Enterprise Knowledge Portal，EKP），或知识门户，其渊源在于知识管理运动。知识管理建立在通过发挥组织的集体知识和经验的杠杆作用来发展或维持企业的竞争优势这样一个假设之上，这些知识大多存于企业员工的心中，且相互孤立。企业知识门户实际上是提供了一个创造、收集和传播企业知识的平台。它结合了普通的企业信息门户的特征和知识管理的目标，这些目标包括对知识进行记录和分类，对信息进行评估，对特定问题专家进行确认和访问，凭借经验、洞察力和交流能力将各种资料连接起来等等。这样，由于企业知识门户将人和信息连接了起来，提供了一个实验企业知识并最大化其价值的载体，因此可以说企业知识门户是第一个将知识管理的理论用于实践的工具。

通过企业知识门户，任何员工都可以实时地与工作团队中的其他成员取得联系，寻找到能够提供帮助的专家或者快速连接到相关的门户。不难看出，企业知识门户的使用对象是企业员工，它的建立和使用可以大大提高企业范围内的知识共享，并由此提高企业员工的工作效率。企业知识门户的好处在于减少了确认问题和解决问题的时间，降低了训练成本和缩短了学习曲线，从而提高了企业生产率。

当然，企业知识门户还应该具有信息搜集、整理、提炼的功能，可以对已有的知识进行分类，建立企业知识库并随时更新知识库的内容。

3）企业应用门户

企业应用门户（Enterprise Application Portal，EAP）是一个用来提高企业的集中贸

易能力、协同能力和信息管理能力的平台。它以商业流程和企业应用为核心，把商业流程中功能不同的应用模块通过门户技术集成在一起，提供了一个企业内部的无缝集成的应用和后端支持系统。从某种意义上说，我们可以把企业应用门户看成是企业信息系统的集成界面，企业员工和合作伙伴可以通过企业应用门户访问相应的应用系统，实现移动办公、进行网上交易等。

典型的通过企业应用门户集成的应用程序和系统包括企业资源计划和旧版本系统（legacy system）以及客户关系管理系统（CRM），供应链管理系统（SCM）和其他要求可以随时随地访问的关键任务（mission-critical）应用程序。由于企业应用门户的以流程为中心的特点，企业应用门户经常包含有工作流的特征。企业应用门户经常使用复杂的安全手段并集成了现有的或新的轻量级目录访问协议或目录结构来保证用户通过不同的系统进行一目了然的访问时的安全。

由于企业应用门户具有的集成应用程序和系统的能力，它对企业具有比企业信息门户或企业知识门户更大的价值，但是，其建立也更为困难、成本更高也更费时间。

4）垂直门户

垂直门户（vortal）是为某一特定的行业服务的，垂直门户传送的信息只属于人们感兴趣的领域。通常，搜索引擎会记录下那些专为诸如医药市场或法律服务等垂直市场（vertical market）设计的网址或只在那些相关市场的网址中查找相关的主题。

企业信息门户、企业知识门户和企业应用门户这三类门户虽然能满足不同应用的需求，但随着企业信息系统复杂程度的增加，越来越多的企业需要能够将以上三类门户有机地整合在一起的通用型企业门户。按照（International Data Corporation，IDC）的定义，通用型的企业门户应该随访问者角色的不同，允许其访问企业内部网上的相应应用和信息资源。除此之外，企业门户还要提供先进的搜索功能、内容聚合能力、目录服务、安全性、应用/过程/数据集成、协作支持、知识获取、前后台业务系统集成等多种功能。给企业员工、客户、合作伙伴、供应商提供一个虚拟的工作场所。

3．企业门户的要素

到目前为止，部署门户是一项开支巨大的工作。强烈影响企业门户运行的成功还是失败的重要因素如下。

（1）计划和设计。门户建设者应首先制定一份商业计划书，描述从项目实施到产生巨大的商业价值的自然进程以及门户如何改善决策支持系统。在考虑门户的设计时，应解决门户的外观和风格、数据组织、内容管理等等。设计者一开始还必须考虑到集成问题。

（2）技术决策。首先需要做的决定是“买还是建”，这是根据包括企业总的 IT 环境、内部专业人员的水平、部署方法、项目范围在内的诸多因素共同决定的。不管门户的基础架构是依靠外购还是由企业自己建立，关键是要考虑到门户要求所有的企业信息源具有普遍的连通性。此外，垂直企业门户和水平企业门户的集成以及多重水平用户的整体

协调也是在一开始就应该解决的问题。

（3）行政人员的支持。企业门户必须得到组织的主要商业伙伴，尤其是消费者的充分支持，这才能实现有效的决策，改善全公司的交流，促进跨部门的合作和得到充足的资金。

（4）限定初始项目的范围。许多组织将考虑先实施起“概念证明”（proof of concept）作用的小型项目或雏形项目，即首先由门户供应商和其他将通过门户提供特定产品和服务的供应商一起建立一个有限范围的门户，以确保计划的设计方案在成功实施后能够真正发挥人们预期的作用。

（5）超过组织物理界限的扩展。此时项目团队中必须包括来自伙伴组织的成员，这样才可以更好地进行协调和交流。

（6）信息组织。信息分类十分重要且具有很强专业性，所以企业在这一领域需要争取得到外部顾问机构的帮助。

（7）内部推广和门户使用范围。在企业门户项目刚开始时，组织所作的一个典型假设就是用户将自动的和自发的聚集到门户周围。分阶段的进行部署允许用户可以逐渐习惯新变化。

（8）门户实施中的社会和心理因素以及这些因素对企业及其门户方案的影响。

（9）用户对门户的持续管理和支持。

4．什么是电子商务

商业活动与 Internet 的结合产生了电子商务。通俗地说，所谓电子商务就是用数字信号在网上开展商务活动，当企业将它的主要业务通过企业内部网、外部网以及 Internet 与企业的职员、客户、供销商以及合作伙伴直接相连时，其中发生的各种活动就是电子商务。

电子商务可以划分为广义和狭义的电子商务。广义的电子商务定义为，使用各种电子工具从事商务或活动。这些工具包括从初级的电报、电话、广播、电视、传真到计算机、计算机网络，到国家信息基础结构－信息高速公路（National Information Infrastructure，NII）、全球信息基础结构（Global Information Infrastructure，GII）和 Internet 等现代系统。而商务活动是从泛商品（实物与非实物，商品与非商品化的生产要素等）的需求活动到泛商品的合理、合法的消费除去典型的生产过程后的所有活动。狭义电子商务定义为，主要利用 Internet 从事商务或活动。

电子商务是在技术、经济高度发达的现代社会里，掌握信息技术和商务规则的人，系统化地运用电子工具，高效率、低成本地从事以商品交换为中心的各种活动的总称。这个分析突出了电子商务的前提、中心、重点、目的和标准，指出它应达到的水平和效果，它是对电子商务更严格和体现时代要求的定义，它从系统的观点出发，强调人在系统中的中心地位，将环境与人、人与工具、人与劳动对象有机地联系起来，用系统的目标、系统的组成来定义电子商务，从而使它具有生产力的性质。

5．电子商务的类型

电子商务按电子商务交易涉及的对象、电子商务交易所涉及的商品内容和进行电子业务的企业所使用的网络类型等对电子商务进行不同的分类。

1）按参与交易的对象分类

按参与电子商务交易涉及的对象分类，电子商务可以分为以下三种类型。

- 企业与消费者之间的电子商务（Business to Customer 即 B-TO-C）。这是消费者利用因特网直接参与经济活动的形式，类同于商业电子化的零售商务。目前，在因特网上有许许多多各种类型的虚拟商店和虚拟企业，提供各种与商品销售有关的服务。
- 企业与企业之间的电子商务（Business to Business 即 B-TO-B）。B-TO-B 方式是电子商务应用最重和最受企业重视的形式，企业可以使用 Internet 或其他网络对每笔交易寻找最佳合作伙伴，完成从定购到结算的全部交易行为。
- 企业与政府方面的电子商务（B-TO-G）。这种商务活动覆盖企业与政府组织间的各项事务。例如企业与政府之间进行的各种手续的报批等。

2）按交易涉及的商品内容分类

按照电子商务交易所涉及的商品内容分类，电子商务主要包括两类商业活动。

- 间接电子商务，电子商务涉及的商品是有形货物的电子订货，如鲜花、书籍、食品和汽车等。
- 直接电子商务，电子商务涉及的商品是无形的货物和服务，如计算机软件、娱乐内容的联机订购、付款和交付，或者是全球规模的信息服务。直接电子商务能使双方越过地理界线直接进行交易，充分挖掘全球市场的潜力。

3）按电子商务使用的网络类型分类

根据开展电子商务业务的企业所使用的网络类型框架的不同，电子商务可以分为如下三种形式。

- EDI（Electronic Data Interchange，电子数据交换）网络电子商务。EDI 是按照一个公认的标准和协议，将商务活动中涉及的文件标准化和格式化，通过计算机网络，在贸易伙伴的计算机网络系统之间进行数据交换和自动处理。EDI 主要应用于企业与企业、企业与批发商、批发商与零售商之间的批发业务。EDI 电子商务在 20 世纪 90 年代已得到较大的发展，技术上也较为成熟，但是因为开展 EDI 对企业有较高的管理、资金和技术的要求，因此至今尚不太普及。
- 因特网电子商务（Internet 网络）。是指利用连通全球的 Internet 网络开展的电子商务活动，在因特网上可以进行各种形式的电子商务业务，所涉及的领域广泛，全世界各个企业和个人都可以参与，是目前电子商务的主要形式。
- 内联网络电子商务（Intranet 网络）。是指在一个大型企业的内部或一个行业内开展的电子商务活动，形成一个商务活动链，可以大大提高工作效率和降低业务的

成本。例如中华人民共和国专利局的主页，客户在该网站上可以查询到有关中国专利的所有信息和业务流程，这是电子商务在政府机关办公事务中的应用。

6．电子商务的标准

1）基于 XML 的电子商务标准

为提高信息交换的效率、借鉴 EDI 标准规范的经验，有关公司、行业协会和国际标准化组织相继推出了一些基于 XML 的电子商务标准框架。这些标准框架的目标都是要通过互联网实现企业间高效、可互操作的信息交换和信息处理，其中比较典型的标准规范有 OBI、IOTP、eCo 框架、BizTalk、RosettaNet、cnXML、xCBL 等。

2000 年，UN/CEFACT 和 OASIS 两个分别代表着传统标准和新兴标准的制定组织共同在全球范围内发起了基于 XML 的电子商务标准框架（即 ebXML 标准）的研制工作，该项工作得到了全世界百余企业的支持和参与。目前，基于 XML 的电子商务标准主要如下。

- ebXML：2001 年 5 月第一批 ebXML 相关标准规范正式发布，ebXML 是全球基于 XML 的电子商务信息交换框架，它向全球各贸易参与方提供一种可互操作的、安全稳定的电子商务信息交换模式。ebXML 是一系列构成电子商务模型框架的技术规范的统称，通过这些技术规范来构建一个全球电子化市场，在这个市场内不分地域和规模的各类企业能够通过交换基于 XML 的电子业务信息开展彼此间的业务。ebXML 力图建立一种基于开放式标准的电子商务理论框架，为电子商务实施提供理论指导。ebXML 是一项庞大复杂的工程项目，它的最终实现还需要大量的基础性标准和相关产品的支持，值得密切关注的是 Web 服务技术的发展。
- RosettaNet：一个致力于开发和实现全行业开放式电子商务流程标准的信息技术、电子元件和半导体制造业企业联合组织，基于 RosettaNet 的 B2B 系统整合有助于加速供应链的协作、增强企业核心竞争力。这个标准有广泛的业界支持，目前全球已有超过 400 家企业采用，而标准的实施成本少于 5 万美元，性价比相当高。英特尔就采用 RosettaNet 标准和 450 多家合作伙伴进行交易。据悉，RosettaNet 在中国将开展一系列标准推广计划，包括与政府部门、领先高科技公司及跨国公司成立电子商务工作小组，召开互操作性大会和泛亚地区峰会等。
- cnXML：中国科学院软件研究所电子商务技术研究中心提出了以国际 XML 标准为基础、与国际其他相关标准可相互转换的、符合我国商业流程习惯与传统的 B2B 电子商务语法、具有中国特色的电子商务信息化规范——cnXML。cnXML 在数据结构上首次提出了中英双语标准的概念，不仅支持英文标签，还全面支持中文标签。在双语的标准的构架下，国内企业在使用 cnXML 规范的时候不仅没有母语的障碍，同时在从事国际交易的时候又不给国外企业造成语言上的新障碍，使各个交易方能够便利地与国内外其他电子商务交易语言进行交互。

2）基于 Web 服务的电子商务集成标准

Web 服务是指由企业发布的完成其特别商务需求的在线应用服务，其他公司或应用软件能够通过互联网来访问并使用这项在线服务。Web 服务的目标是将软件转化为一种通过 Web 订阅使用的服务。在 Web 服务模式下，软件将运行于中央 Web 服务器、而非用户的 PC 中。这样，从理论上来说，用户就能够通过 PC、移动电话、掌上电脑或任一接入互联网的设备访问各种类型的应用与服务，并能够自动实现应用与服务的实时更新与升级。Web 服务模式的核心是能够实现更简便的、基于 XML 的在线数据交换。

微软、IBM、Sun、Oracle 及其他有关厂商纷纷摈弃了各自不同的技术标准，共同选定了万维网联盟（World Wide Web Consortium）、简单对象访问协议（Simple Object Access Protocol）、互联网服务描述语言（Web Services Description Language）和统一显示接口（Unified Display Interface）四种基于 XML 的相关标准作为 Web 服务的底层架构技术。另外，在 W3C 联盟及 OASIS 等业内标准组织的协助下，微软、IBM 等公司还计划进一步合作，共同制定对全球 Web 服务市场发展至关重要的诸如安全与可靠性等方面的 Web 服务标准。

3.3.6　企业应用集成

1．企业应用集成（EAI）的简要历史

20 世纪 50 年代末到 60 年代初，企业具备了早期应用，这些应用大多是用来替代重复性劳动的一些简单设计。当时企业应用唯一的目标就是用计算机代替一些孤立的、体力性质的工作环节。

20 世纪 60 年代中期，继第一代应用之后，更深入辅助企业生产的应用出现，诸如库存管理、生产控制和早期的财务管理等。这些应用的思想还仅仅是支持企业业务的一部分，并且企业数据的访问控制技术还不完善，仍然没有企业数据集成的概念。

20 世纪 60 年代末到 70 年代初，数据库的技术开始出现，磁盘存储和数据库技术使企业能够直接访问数据，这让企业应用可进行在线联机处理。在线联机应用是根据局部需求开发的，在不同的部门或企业间开始出现早期电子数据交换（EDI），即企业数据集成的概念开始出现。但这些数据集成还仅仅是小范围的小火花，距离企业应用集成还很遥远。

20 世纪 80 年代，企业应用开始不能满足企业新的需求，很多公司的技术人员都试图在企业系统整体概念的指导下对已经存在的应用进行重新设计，以便让它们集成在一起。然而这种努力收效甚微。80 年代中期，C/S 结构的应用开始出现。用于处理 C/S 结构下联机事务处理的中间件（TUXEDO）出现。企业应用集成软件本身就是一种中间件的技术，当中间件技术出现，企业应用集成具备了发展的可能——思想上的基础。这段时间诞生了信息总线软件，信息总线软件提供了企业应用集成最基础的内容——信息交换。这一层次的 EAI 的系统集成框架，主要解决的是企业内部应用系统间的信息共享的

问题。解决方案的构建的出发点是整体考虑企业应用系统 IT 建设，统一建设信息交换基础，消除了点对点集成企业应用的混乱局面。

20 世纪 90 年代，ERP 应用开始流行的时候，同时也要求它们能够支持已经存在的应用和数据，这就必须引入 EAI（Enterprise Application Integration，系统应用集成）。对 EAI 的需求首先来自于企业将它们的主机系统转换成 C/S 结构系统的过程中，其次是利用 ERP 建立企业骨干信息系统时。企业迫切需要一种方法，让它们少写程序，无须花巨大的费用，就可以将各种旧的应用系统和新的系统集成起来。其他推动 EAI 市场的因素还有供应链管理（B2B 集成）、基于流程的业务处理以及 Web 应用集成。随着企业各种应用的迅速增加以及更多地把自己的业务转向电子商务，EAI 方案对企业的重要性也日益显现。越来越多的企业开始采用 EAI 解决方案将企业内部的应用软件与外部客户和供应商的应用软件进行链接，实现数据流和业务运作的自动化，从而达到业务的实时与快速。EAI 与电子商务的结合为企业快速实现业务的自动化提供了可靠的保证，呈现在我们面前的将是一个同时具有数据自动化和业务流程高度可塑的企业管理框架，从而进一步加快端到端的电子商务应用集成，包括供应链管理、客户关系管理和 ERP 系统相关联的门户网站、前端应用、后端应用等。

2．EAI 的内容

EAI 构建统一标准的基础平台，将进程、软件、标准和硬件联合起来，连接具有不同功能和目的而又独自运行的企业内部的应用系统，以达到信息和流程的共享,使企业相关应用整合在一起。

EAI 就是在各个应用系统的接口之间共享数据和功能。EAI 的一个原则就是集成多个系统并保证各个系统互不干扰。EAI 的终极目标就是将多个企业和企业内部的多个应用集成到一个虚拟的、统一的应用系统中。 因此，实施 EAI 必须遵循如下原则：应用程序的独立性；面向商业流程；独立于技术；平台无关。

EAI 提供 4 个层次的服务,从下至上依次为通讯服务、信息传递与转化服务、应用连接服务、流程控制服务。

通讯服务主要靠通讯中间件进行消息的路由；信息传递与转化服务主要负责传递消息和转化消息；应用连接服务主要靠应用连接适配器将应用连接至 EAI 平台，最终连接起来；流程控制服务解决人工参与的长期的工作流程控制。

从集成的深度上来说，从易到难有以下种类的集成：

（1）数据的集成。

（2）应用系统的集成。

（3）业务流程的集成。

从集成的广度上来看，从易到难有以下种类的集成：

（1）部门内部的信息系统集成。

（2）部门之间的信息系统集成。

（3）企业级的信息系统集成。

（4）与有稳定关系的合作伙伴之间的信息系统实现面向业务过程的集成。

（5）与随机遇到的合作伙伴之间的信息系统实现面向业务过程集成。

3．集成技术的发展展望

目前市场主流的集成模式有三种，分别是面向信息的集成技术、面向过程的集成技术和面向服务的集成技术。

在数据集成的层面上，信息集成技术仍然是必选的方法。信息集成采用的主要数据处理技术有数据复制、数据聚合和接口集成等。其中，接口集成仍然是一种主流技术。它通过一种集成代理的方式实现集成，即为应用系统创建适配器作为自己的代理，适配器通过其开放或私有接口将信息从应用系统中提取出来，并通过开放接口与外界系统实现信息交互，而假如适配器的结构支持一定的标准，则将极大地简化集成的复杂度，并有助于标准化，这也是面向接口集成方法的主要优势来源。标准化的适配器技术可以使企业从第三方供应商获取适配器，从而使集成技术简单化。

面向过程的集成技术其实是一种过程流集成的思想，它不需要处理用户界面开发、数据库逻辑、事务逻辑等，而只是处理系统之间的过程逻辑，与核心业务逻辑相分离。在结构上，面向过程的集成方法在面向接口的集成方案之上，定义了另外的过程逻辑层；而在该结构的底层，应用服务器、消息中间件提供了支持数据传输和跨过程协调的基础服务。对于提供集成代理、消息中间件以及应用服务器的厂商来说，提供用于业务过程集成是对其产品的重要拓展，也是目前应用集成市场的重要需求。

基于 Service-Oriented Arditecture（面向服务架构）和 Web 服务技术的应用集成是业务集成技术上的一次重要的变化，被认为是新一代的应用集成技术。集成的对象是一个个的 Web 服务或者是封装成 Web 服务的业务处理。Web 服务技术由于是基于最广为接受的、开放的技术标准（如 HTTP、SMTP 等），支持服务接口描述和服务处理的分离、服务描述的集中化存储和发布、服务的自动查找和动态绑定以及服务的组合，成为新一代面向服务的应用系统的构建和应用系统集成的基础设施。

3.3.7　供应链管理

研究报告指出：在市场变化加快、全球化竞争日益激烈的情况下，单个企业仅仅依靠自己内部资源的整合已难以满足快速变化的市场需求。“横向一体化”是解决该问题的一个途径，其思想是：企业将有限的资源集中于自己的核心业务，并与其他企业建立合作伙伴关系，通过不同企业之间的分工，进行优势互补以获得集体竞争优势，提高整体竞争力，达到双赢的效果。由此产生一种横向一体化的企业管理模式——供应链管理。

1．供应链管理的定义

所谓供应链，是指产品生产和流通过程中涉及的原材料供应商、制造商、批发商、零售商以及最终消费者组成的供需网络。供应链是企业赖以生存的商业循环系统，是企

业电子商务中最重要的要素。统计数据表明，企业供应链可以耗费企业高达25%的运营成本。

供应链管理是指对供应商、制造商、物流者和分销商等各种经济活动，有效开展集成管理，以正确的数量和质量，正确的地点，正确的时间，进行产品制造和分销，提高系统效率，促使系统成本最小化，并提高消费者的满意度和服务水准。

2．供应链管理的基本思想

随着因特网的普及，物流管理很自然地上升为供应链管理。因为在整个交易过程中可能会存在一些矛盾和冲突，供应链管理可以弥合整个体系中的矛盾和冲突。供应链管理是一种集成化的管理模式。它是一种从供应商开始，经由制造商、分销商、零售商，直到最终客户的全要素、全过程的集成化管理模式，是一种新的管理策略，它把不同的企业集成起来以增加整个供应链的效率，注重的是企业之间的合作，以达到全局最优。通过供应链管理，处于供应链上的各个企业明确各自在整个体系中所处的角色，协调好相互之间的关系，企业之间建立起有效的信息共享机制，使供应链中的信息流、物流和资金流类似于一个整体运作，能快速满足最终客户不断变化的需求。

3．供应链管理的运作模式

供应链中的信息流覆盖了从供应商、制造商到分销商，再到零售商等供应链中的所有环节。当需求信息（如客户订单、生产计划、采购合同等）从需方向供方流动时，便引发物流。供应链的基本运作模式可以分为两种，如果把供应链中物流的方向确定为供应链的方向，那么供应链的两种运作模式可以分别称为正向推动式运作模式和逆向拉动式运作模式。

正向推动式运作模式的推动力来自供应链的上游企业，适合于市场需求量很大而且需求稳定的通用型产品，其指导思想是“以生产为中心”。在当今的技术条件下，从产品的功能上讲，能够提供同样产品的企业会越来越多，因而正向推动式运作模式适用的范围也会越来越窄。用发展的眼光来观察，正向推动式运作模式柔性较差，快速响应能力不强，不是供应链普遍适用的运作模式。

基于需求驱动原理的供应链运作模式是一种逆向拉动式运作模式，驱动力来源于最终用户，与正向推动式运作模式有着本质的区别。正向推动式运作模式是以生产为中心，而逆向拉动式运作模式是以用户为中心。两种不同的运作模式分别适用于不同的市场环境，有着不同的运作效果。逆向拉动模式反映的是经营理念从“以生产为中心”向“以顾客为中心的”的转变。来源于市场需求和供应链下游企业的订单，通过集成化供应链信息系统快速逐级向上驱动，使得供应链系统能够准时响应市场需求，物流速度得以提高，供应链总成本下降，快速反应能力增强，最终体现在供应链系统的综合竞争能力提高。

处于供应链核心环节的企业要将与自己业务有关（直接和间接）的上下游企业纳入一条环环相扣的供应链中，使多个企业能在一个整体的信息系统管理下实现协作经营和

协调运作，把这些企业的分散计划纳入整个供应链的计划中，实现资源和信息共享，增强了该供应链在市场中的整体优势，同时也使每个企业均可实现以最小的个别成本和转换成本来获得成本优势。这种网络化的企业运作模式拆除了企业的围墙，将各个企业独立的信息孤岛连接在一起，通过网络、电子商务把过去分离的业务过程集成起来，覆盖了从供应商到客户的全部过程。对供应链中的企业进行流程再造，建立网络化的企业运作模式是建立企业间的供应链信息共享系统的基石。

4. 供应链管理的技术支持体系

为了实现企业的目标，必须通过信息的不断传递，一方面进行纵向的上下信息传递，把不同层次的经济行为协调起来；另一方面进行横向的信息传递，把各部门、各岗位的经济行为协调起来。此时，供应链信息系统需要大量的信息技术来支持。

信息技术对供应链的支撑可以分为两个层面。

第一个层面是由核心信息技术构成，主要有以下几种：

（1）标识代码技术。对大量的信息进行合理的分类后或者为了对编码对象进行唯一表示而用代码加以表示。信息编码的标准化可以实现供应链中贸易伙伴间的数据交换与共享。

（2）自动识别与数据采集技术（Automated Identification and Data Collection，AIDC）。通过自动识别项目标识信息，并且不使用键盘即可将数据直接输入到计算机、程序逻辑控制器或其他微处理器控制设备。AIDC 技术包括条码技术、射频技术、磁识别技术、声音识别技术、图像识别技术、光字符识别技术、生物识别技术和空间数据传输技术。

（3）电子数据交换技术（EDI）。

EDI 是供应链管理的主要信息手段之一，特别是在国际贸易中有大量文件传输的条件下，它是计算机与计算机之间的相关业务数据的交换工具，它有一致的标准以使交换成为可能。

（4）互联网技术。互联网可以使人们快速访问众多的资源，也可以进行网上交易活动。供应链成员可以通过互联网及时地获得供应链上的有关信息，互联网为供应链信息共享提供了一个基础的工具。

第二层面是基于信息技术而开发的支持企业生产的管理系统。

在具体集成和应用这些系统时，不应仅仅将它们视为是一种技术解决方案，而应深刻理解它们所折射的管理思想。

（1）销售时点信息系统（Point of Sale，POS）。POS 是指通过自动读取设备（收银机）在销售商品时直接读取商品销售信息，并通过网络和计算机系统传送至有关部门进行分析加工，以提高经营效率的系统。

（2）电子自动订货系统（EOS）。EOS 是指企业间利用网络（VAN 或 Internet）和终端设备以在线（ON-LINE）方式进行订货作业和订货信息交换的系统。相对于传统的订货方式，EOS 系统可以缩短从接到订单到发出订货的时间，缩短订货商品的交货期，减

少商品订单的出错率；有利于减少企业的库存水平，提高企业的库存管理效率；对于生产厂家和批发商来说，通过分析零售商的商品订货信息，能准确判断畅销商品和滞销商品，有利于调整商品生产和销售计划。

（3）计算机辅助设计（Computer-Aided Process Planning，CAD）、计算机辅助工艺规划（Computer-Aided Process Plaming，CAPP）、计算机辅助工程（CAE）和计算机辅助制造（Computer-Aided Process Plaming，CAM）等计算机辅助技术主要用于支持新产品设计与制造。随着产品数据管理（Product Data Management，PDM）的发展，有效地建立了 CAD、CAPP、CAE、CAM 之间的信息集成，实现供应链上各企业之间正确而快速的数据交换，从而进一步加快产品开发时间，降低了费用。

（4）企业资源计划（ERP）、制造资源计划（MRPII）、及时生产制（JIT）。ERP、MRPII、JIT 等主要用于企业生产控制和库存控制。当然 ERP 的范围更广，已体现出了供应链管理的思想，其应用领域从传统制造业拓展到其他类型的行业。ERP、 MRPII、JIT 等技术的应用可以解决企业生产中出现的多种复杂问题，促进了企业业务流程、信息流程和组织结构的变革，提高企业生产和整个供应链的柔性，保证生产及供应链的正常运行。

（5）客户关系管理（CRM）。CRM 最主要的功能模块是客户服务、市场营销、销售。通过将 CRM 应用于企业之间的信息共享，可以提升供应链上各企业之间的服务水平，提高客户满意度，维持较高的客户保留，对客户收益和潜在收益产生积极的影响等。

（6）电子商务。电子商务是各参与方之间以电子方式而不是通过物理交换或直接物理接触完成的任何形式的业务交易，它包括电子数据交换、电子支付手段、电子订货系统、电子邮件、传真、网络、电子公告系统、条码、图像处理、智能卡等。在供应链管理中，电子商务一般为企业对企业（B2B）和企业对消费者（B2C）两种类型。电子商务在供货体系管理、库存管理、运输管理和信息流通等方面提高了企业供应链管理运作的效率。

3.3.8 信息化的有关法律和规定

1．有关信息产业发展的法律和规定

1）电信条例

电信产业是国家信息化的支柱型产业，是信息化的公共网络和平台。2000 年 9 月 25 日由国务院公布施行的《中华人民共和国电信条例》，它的宗旨是“规范电信市场秩序，维护电信用户和电信业务经营者的合法权益，保障电信网络和信息的安全，促进电信业的健康发展”。“电信条例”是信息化、计算机信息网络管理和服务管理的重要规定。

2）国务院 2008 年第 18 号文件

国务院于 2000 年 6 月 25 日颁布了《鼓励软件产业和集成电路产业发展的若干政策》即“18 号文件”，这是第一个鼓励和支持软件产业发展的专项产业政策，也是我国软件产业发展史上的重要里程碑。

“18 号文件”明确规定了“政策目标”：一是“通过政策引导，鼓励资金、人才等资源投向软件产业和集成电路产业，进一步促进我国信息产业快速发展，力争到 2010 年使我国软件产业研究开发和生产能力达到或接近国际先进水平。”二是“鼓励国内企业充分利用国际、国内两种资源，努力开拓两个市场。经过 5～10 年的努力，国产软件产品能够满足国内市场大部分需求，并有大量出口；国产集成电路产品能够满足国内市场大部分需求，并有一定数量的出口，同时进一步缩小与发达国家在开发和生产技术上的差距。”

18 号文件为推动我国软件产业和集成电路产业的发展制定了一系列如投融资政策、税收政策、产业技术政策、出口政策、收入分配政策、人才吸引与培养政策、采购政策、软件企业认定制度、知识产权保护、行业组织、行业管理和集成电路产业政策等。

3）信息产业主管部门的有关规定

（1）软件企业认定制度。

为了加速我国软件产业的发展，增强信息产业创新能力和国际竞争力，根据国务院《鼓励软件产业和集成电路产业发展的若干政策》，信息产业部、教育部、科学技术部、国家税务总局于 2000 年 10 月制定并颁布了《软件企业认定标准及管理办法（试行）》。根据该办法，信息产业部对全国软件产业实行行业管理和监督，组织协调并管理全国软件企业认定工作，其主要职责是：根据各省、自治区、直辖市信息产业主管部门的建议，确定各地省级软件企业认定机构，向其授权或撤销对其授权，并公布软件企业认定机构名单；指导并监督、检查全国软件企业认定工作；受理对认定结果和年审结果的复审申请。各省、自治区、直辖市信息产业主管部门负责管理本行政区域内的软件企业认定工作，其主要职责与信息产业部相类同。

（2）软件产品管理办法。

为了加强软件产品管理，促进我国软件产业的发展，根据国家有关法律法规和国务院《鼓励软件产业和集成电路产业发展的若干政策》，2001 年 6 月信息产业部以第 5 号令的形式，颁布了《软件产品管理办法》。“办法”规定，“软件产品的开发、生产、销售、进出口等活动应遵守我国有关法律、法规和标准规范。任何单位和个人不得开发、生产、销售、进出口含有以下内容的软件产品：（一）侵犯他人的知识产权的；（二）含有计算机病毒的；（三）可能危害计算机系统安全的；（四）含有国家规定禁止传播的内容的；（五）不符合我国软件标准规范的”。

“办法”同时规定，信息产业部负责全国软件产品的管理。其主要职责是：制定并发布软件产品测试标准和规范；对各省、自治区、直辖市登记的国产软件产品备案；指导并监督、检查全国各地的软件产品管理工作；授权软件产品检测机构，按照我国软件产品的标准规范和软件产品的测试标准及规范，进行符合性检测；制定全国统一的软件产品登记号码体系、制作软件产品登记证书；发布软件产品登记通告。“办法”还规定了各省、自治区、直辖市信息产业主管部门负责本行政区域内软件产品的管理工作。

2．有关知识产权保护的法律和规定

知识产权保护方面的法律和规定是我国法律体系的重要组成部分，主要包括专利法、商标法和著作权法等。由于计算机软件既是一种著作，属于“著作权法”的保护范畴，但计算机软件又不同于一般著作，因此，为了保护计算机软件著作权人的权益，调整计算机软件在开发、传播和使用中发生的利益关系，鼓励计算机软件的开发与应用，促进软件产业和国民经济信息化的发展，根据《中华人民共和国著作权法》，国务院于2001年12月颁布了新修订的《计算机软件保护条例》。

《计算机软件保护条例》对计算机软件著作权的权利、归属以及侵权行为的鉴别做了详细的规定，并指出侵权行为应负的法律责任。《计算机软件保护条例》为软件的版权保护提供法律依据，有利于软件业的健康发展。

3．计算机信息网络管理的法律和规定

1）《信息网络传播权保护条例》

《信息网络传播权保护条例》进一步完善、健全了对信息网络传播权的保护制度。此前，我国已建立信息网络传播权的保护制度，并出台了相关法律、行政法规、司法解释及规章。2000年，最高人民法院制定了《关于审理涉及计算机网络著作权纠纷案件适用法律若干问题的解释》；2001年，全国人大常委会修订了《中华人民共和国著作权法》；2002年，国务院颁布了《中华人民共和国著作权法实施条例》；2003年，最高人民法院修订了《关于审理涉及计算机网络著作权纠纷案件适用法律若干问题的解释》；2005年，国家版权局和信息产业部联合发布了《互联网著作权行政保护办法》等。通过上述法律、法规及相关司法解释可以看出，侵犯他人信息网络传播权不仅要承担民事责任，还要承担行政责任，甚至是刑事责任（构成犯罪的）。作为互联网信息网络服务的主要提供者，基础电信运营企业在提供信息网络服务过程中应树立保护信息网络传播权的观念，包括保护自身权益和尊重他人权益的意识。

《信息网络传播权保护条例》明确规定了为网络服务提供者提供“避风港”。“避风港”条款是指在发生著作权侵权案件时，当ISP（网络服务提供商）只提供空间服务，并不制作网页内容，如果ISP被告知侵权，则有删除的义务，否则就被视为侵权。如果侵权内容既不在ISP的服务器上存储，又没有被告知哪些内容应该删除，则ISP不承担侵权责任。“避风港”条款也被扩展应用于提供搜索引擎、网络存储、在线图书馆等服务的提供商。

2）《中华人民共和国计算机信息网络国际联网管理暂行规定实施办法》

1997年12月8日国务院信息化工作领导小组审定《中华人民共和国计算机信息网络国际联网管理暂行规定实施办法》，1998年3月6日发布，旨在加强对计算机信息网络国际联网的管理，保障国际计算机信息流的健康发展。该实施办法规定国家对国际联网的建设布局、资源利用进行统筹规划，对国际出入口信道统一管理。

3）《计算机信息网络国际联网安全保护管理办法》和《互联网安全保护技术措施规定》

2005 年中华人民共和国公安部第 82 号令发布的《互联网安全保护技术措施规定》2006 年 3 月 1 日起在全国实施。根据该规定，互联网服务提供者、联网使用单位负责落实互联网安全保护技术措施，并保障互联网安全保护技术措施功能的正常发挥。公安部于 1997 年 12 月 30 日发布施行了《计算机信息网络国际联网安全保护管理办法》，该规定为了保护计算机信息系统的安全，促进计算机的应用和发展。《互联网安全保护技术措施规定》是一部与《办法》相配套的规章，对互联网服务单位和联网使用单位落实安全保护技术措施提出了明确和具体的要求。

《互联网安全保护技术措施规定》从保障和促进我国互联网发展出发，对互联网服务单位和联网单位落实安全保护技术措施提出了明确、具体和可操作的要求，有利于加强和规范互联网安全保护工作，提高互联网服务单位和联网单位的安全防范能力和水平，预防和制止网上违法犯罪活动。从 2006 年 3 月 1 日起，公安机关将依法对辖区内互联网服务提供者和联网使用单位安全保护技术措施的落实情况进行指导、监督和检查。对违反《互联网安全保护技术措施规定》的，将依照《计算机信息网络国际联网安全保护管理办法》的有关规定予以处罚。

第 4 章　系统开发基础知识

4.1　软件开发方法

4.1.1　软件开发生命周期

传统的软件生命期（software life cycle）是指软件产品从形成概念（构思）开始，经过定义、开发、使用和维护，直到最后被废弃（不能再使用）为止的全过程。按照传统的软件生命周期方法学，可以把软件生命期划分为软件定义、软件开发、软件运行与维护三个阶段。

1．软件定义时期

软件定义包括可行性研究和详细需求分析过程，任务是确定软件开发工程必须完成的总目标。具体可分成问题定义、可行性研究、需求分析等。

（1）问题定义。问题定义是人们常说的软件的目标系统是“什么”，系统的定位以及范围等。也就是要按照软件系统工程需求来确定问题空间的性质（说明是一种什么性质的系统）。

（2）可行性研究。软件系统的可行性研究包括技术可行性、经济可行性、操作可行性和社会可行性等，确定问题是否有解，解决办法是否可行。

（3）需求分析。需求分析的任务是确定软件系统的功能需求、性能需求和运行环境的约束，写出软件需求规格说明书、软件系统测试大纲、用户手册概要。功能需求是软件必须完成的功能；性能需求是软件的安全性、可靠性、可维护性、结果的精度、容错性（出错处理）、响应速度和适应性等；运行环境是软件必须满足运行环境的要求，包括硬件和软件平台。

需求分析是重要的，然而又是困难的。作为开发者，要充分理解用户的需求，并以书面形式写出规格说明书，这是以后软件设计和验收的依据；困难的地方是，由于软件系统的复杂性，作为用户也许很难一次性说清楚系统应该做什么。因此，需求分析也就十分艰巨，它要完成大量的工作。

需求分析过程应该由系统分析员、软件开发人员与用户共同完成，反复讨论和协商，并且逐步细化、一致化、完全化等，直至建立一个完整的分析模型。需求分析工作完成后要提交软件需求规格说明（Software Requirements Specification，SRS）。内容可以有系统（或子系统）名称、功能描述、接口、基本数据结构、性能、设计需求、开发标准、

验收原则等。

2．软件开发时期

软件开发时期就是软件的设计与实现，可分成概要（总体）设计、详细设计、编码、测试等。

概要设计是在软件需求规格说明的基础上，建立系统的总体结构（含子系统的划分）和模块间的关系，定义功能模块及各功能模块之间的关系。

详细设计对概要设计产生的功能模块逐步细化，把模块内部细节转化为可编程的程序过程性描述。详细设计包括算法与数据结构、数据分布、数据组织、模块间接口信息和用户界面等的设计，并写出详细设计报告。

编码又称编程，编码的任务是把详细设计转化为能在计算机上运行的程序。测试可分成单元测试、集成测试、确认测试和系统测试等。通常把编码和测试称为系统的实现。

3．软件运行和维护

软件运行就是把软件产品移交给用户使用。软件投入运行后的主要任务是使软件持久满足用户的要求。

软件维护是对软件产品进行修改或对软件需求变化做出响应的过程，也就是尽可能地延长软件的寿命。

当软件已没有维护的价值时，宣告退役，软件生命随之宣告结束。

4.1.2　软件开发模型

软件生存周期模型又称软件开发模型（software develop model）或软件过程模型（software process model），它是从某一个特定角度提出的软件过程的简化描述。模型的主要特点是简单化。软件过程模型是软件开发实际过程的抽象与概括，它应该包括构成软件过程的各种活动，也就是对软件开发过程各阶段之间关系的一个描述和表示。

软件过程模型的基本概念：软件过程是制作软件产品的一组活动以及结果，这些活动主要由软件人员来完成，软件活动主要如下一些。

（1）软件描述。必须定义软件功能以及使用的限制。

（2）软件开发。也就是软件的设计和实现，软件工程人员制作出能满足描述的软件。

（3）软件有效性验证。软件必须经过严格的验证，以保证能够满足客户的需求。

（4）软件进化。软件随着客户需求的变化不断改进。

软件过程模型是软件工程的重要内容，它为软件工程管理提供里程碑和进度表，为软件开发过程提供原则和方法。软件过程有各种各样的模型，如瀑布模型、演化模型、原型模型、螺旋模型、喷泉模型和基于可重用构件的模型等。软件开发模型至今仍在不断发展和改进，以下介绍主要的几种。

1．瀑布模型

瀑布模型（waterfall model）可以说是最早使用的软件生存周期模型之一。由于这个

模型描述了软件生命的一些基本过程活动，所以它称为软件生命周期模型。这些活动从一个阶段到另一个阶段逐次下降，它的工作流程形式上又很像瀑布，因此人们更常把它称为瀑布模型，该模型如图 4-1 所示。

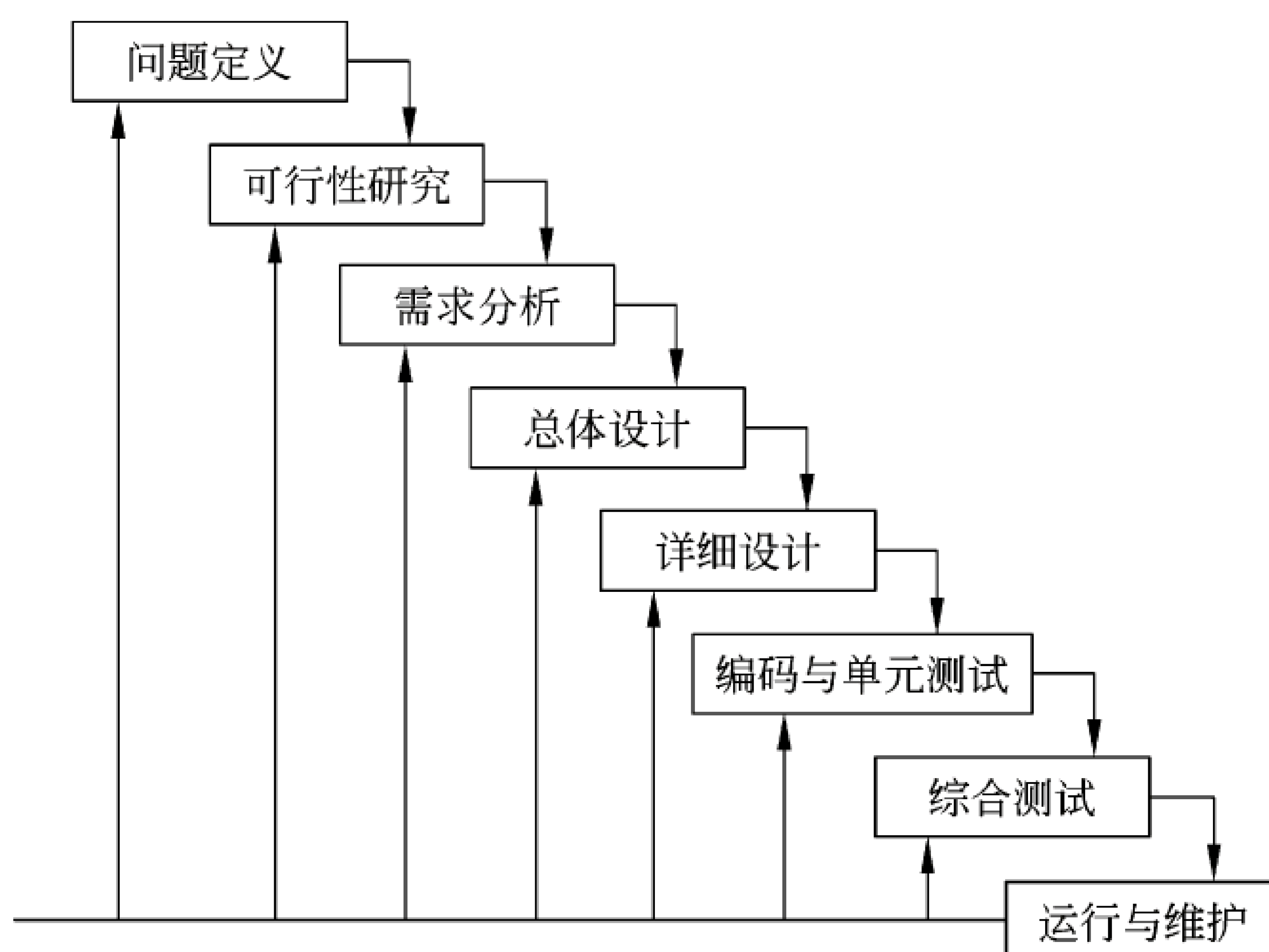


图 4-1　瀑布模型

瀑布模型的特点是因果关系紧密相连，前一个阶段工作的结果是后一个阶段工作的输入。或者说，每一个阶段都是建筑在前一个阶段正确结果之上，前一个阶段的错漏会隐蔽地带到后一个阶段。这种错误有时甚至可能是灾难性的。因此每一个阶段工作完成后，都要进行审查和确认，这是非常重要的。历史上，瀑布模型起到了重要作用，它的出现有利于人员的组织管理，有利于软件开发方法和工具的研究。

瀑布模型的主要缺如下。

（1）软件需求分析的准确性很难确定，甚至是不可能和不现实的。因为用户不理解计算机，无法回答目标系统是“什么”的情况，对系统将来的改变部分难以确定，往往用“我不能准确地告诉你”回答开发人员。

（2）用户和软件项目负责人要相当长的时间才能得到初始版本，这时如果改变需求，将会带来巨大的损失（例如人力、财力、时间等）。该模型的应用有一定的局限性。

2．原型模型

原型模型（prototype model）又称快速原型。由于瀑布型的缺点，人们借鉴建筑师、工程师建造原型的经验，提出了原型模型。该模型如图 4-2 所示。原型模型主要有以下两个阶段：

（1）原型开发阶段。软件开发人员根据用户提出的软件系统的定义，快速地开发一个原型。该原型应该包含目标系统的关键问题和反映目标系统的大致面貌，展示目标系统的全部或部分功能、性能等。

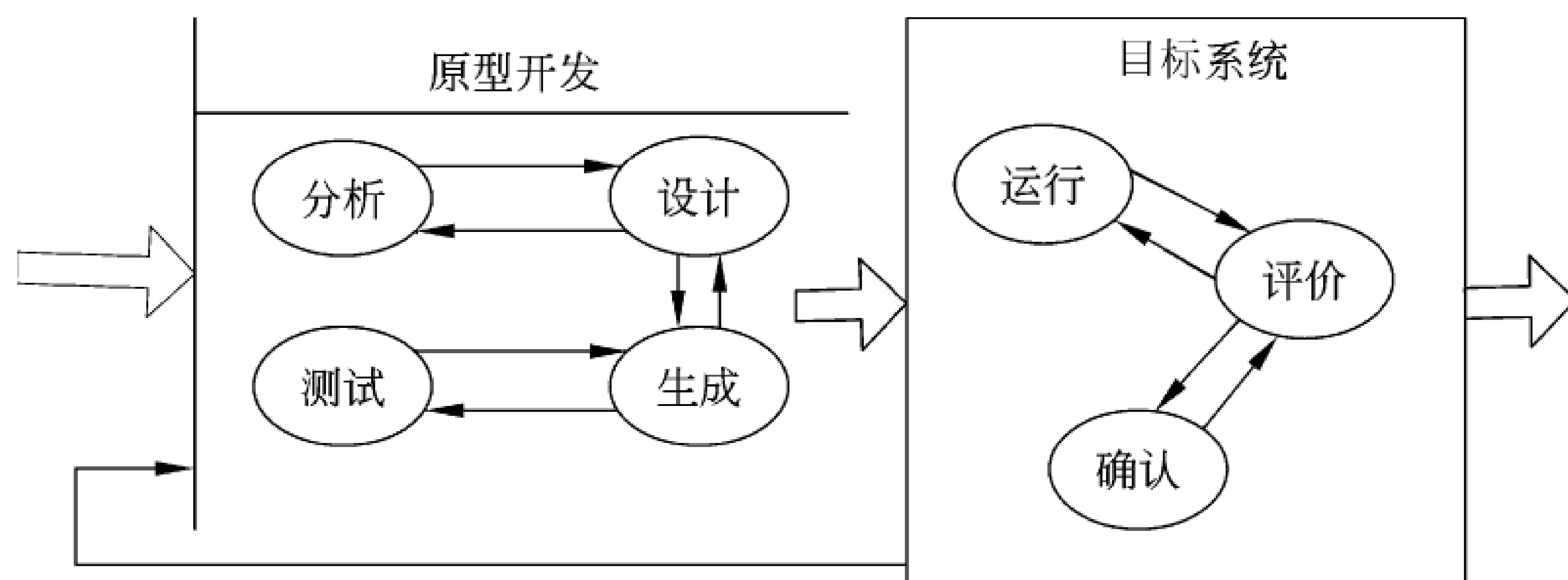


图 4-2　原型模型

开发原型可以考虑以下三种途径：

- 利用模拟软件系统的人机界面和人机交互方式。
- 真正开发一个原型。
- 找来一个或几个正在运行的类似软件进行比较。

（2）目标软件开发阶段。在征求用户对原型的意见后对原型进行修改完善，确认软件系统的需求并达到一致的理解，进一步开发实际系统。但是在实际工作中，由于各种原因，大多数原型都废弃不用，仅仅把建立原型的过程当作帮助定义软件需要的一种手段。原型模型的使用应该注意：

- 用户对系统模糊不清，无法准确回答目标系统的需求。
- 要有一定的开发环境和工具支持。
- 经过对原型的若干次修改，应收敛到目标范围内，否则可能会失败。
- 对大型软件来说，原型可能非常复杂而难以快速形成，如果没有现成的，就不应考虑用原型法。

3．螺旋模型

螺旋模型（Spiral Model）是在快速原型的基础上扩展而成。也有人把螺旋模型归到快速原型，实际上，它是生命周期模型与原型模型的一个结合，如图 4-3 所示。这种模型把整个软件开发流程分成多个阶段，每一个阶段都由 4 部分组成，它们是：

（1）目标设定。为该项目进行需求分析，定义和确定这一个阶段的专门目标，指定对过程和产品的约束，并且制定详细的管理计划。

（2）风险分析。对可选方案进行风险识别和详细分析，制定解决办法，采取有效的措施避免这些风险。

（3）开发和有效性验证。风险评估后，可以为系统选择开发模型，并且进行原型开发，即开发软件产品。

（4）评审。对项目进行评审，以确定是否需要进入螺旋线的下一次回路，如果决定继续，就要制定下一阶段计划。

螺旋模型的软件开发过程实际是上述 4 个部分的迭代过程，每迭代一次，螺旋线就

增加一周，软件系统就生成一个新版本，这个新版本实际上是对目标系统的一个逼近（更接近一步）。经过若干次的迭代后，系统应该尽快地收敛到用户允许或可以接受的目标范围内，否则也有可能中途夭折。

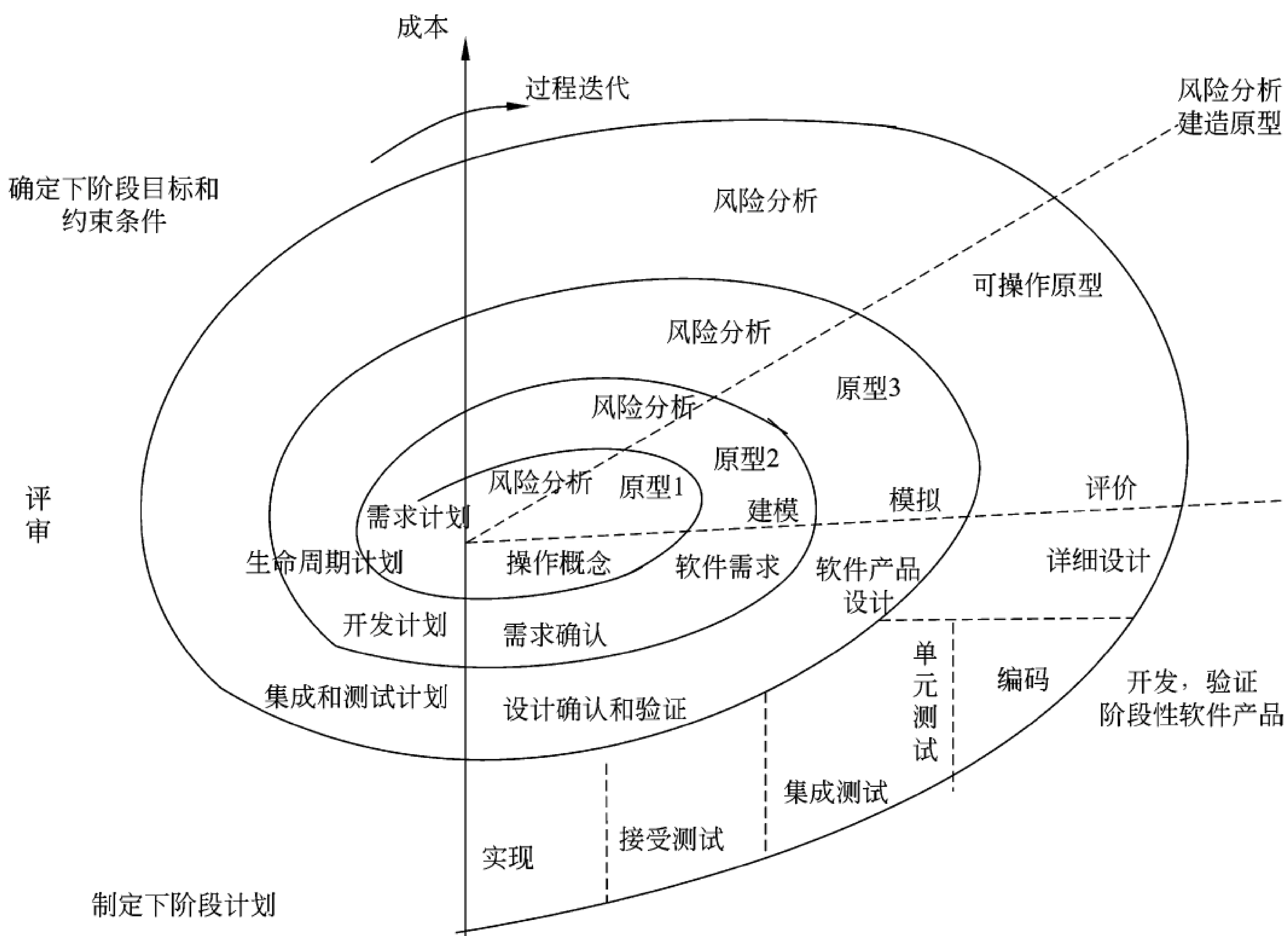


图 4-3　螺旋模型

该模型支持大型软件开发，适用于面向规格说明、面向过程和面向对象的软件开发方法，也适用于几种开发方法的组合。

4．基于可重用构件的模型

事实上，在许多软件项目中都存在一些重用软件。当软件人员意识到某一个项目中的设计或者代码在另一个项目中重复出现时，重用就自然地发生了。构件也正是基于这一思想而产生的。构件的重用提高了软件的可靠性和易维护性，程序在进行修改时产生较少的副作用。近几年来，出现了以组件为基础的软件工程方法，基于构件组装的软件过程模型也随之产生，并且逐渐地被广泛使用。该模型如图 4-4 所示。

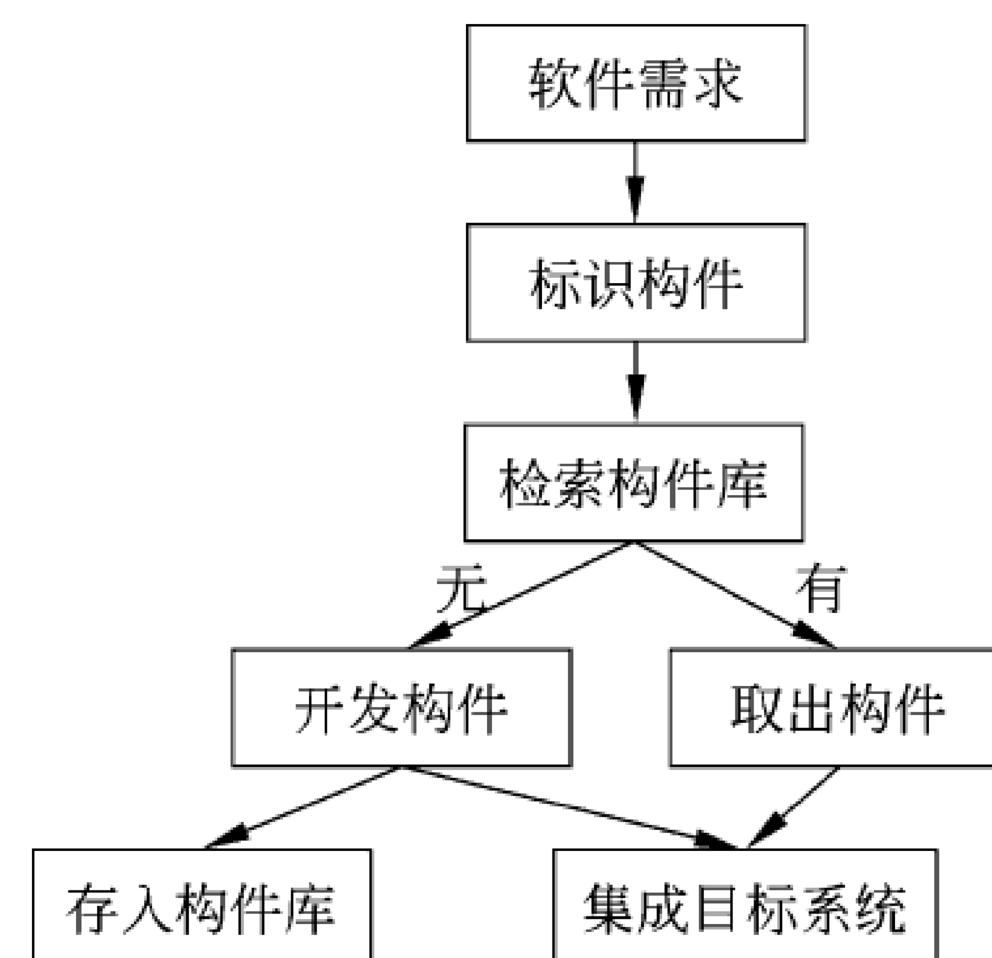


图 4-4　基于可重用构件的模型

人们可以把软件工程项目所创建的构件不断地积

累和存储在一个构件库中，在一个系统开发过程中，一旦标识出候选构件，则可以在构件库中检索该构件，确认这些构件是否存在，如果构件已存在，就可以从构件库中取出重用。如果一个候选构件在构件库中并不存在，那么，就要进行新构件的开发。新构件开发成功后，一方面用它来构造目标系统，另一方面可以把它存入构件库中。软件目标系统是基于可重用构件的一种集成，这将大大地提高软件的可靠性和生产率。

显然，一个系统将依赖构件的健壮性。但毫无疑问，构件组装模型使软件可以重用，而重用给软件工程师提供大量的好处。构件组装模型具有极其广阔的实用性和深远的意义。

5．基于面向对象的模型

面向对象技术自从问世后，很快被人们所接受，并得到广泛的应用。面向对象技术确实有很多的优点，其中构件重用是非常重要的技术之一。对象技术强调了类的创建与封装，一旦一个类创建与封装成功，就可以在不同的应用系统中被重用。

对象技术为基于构件的软件过程模型提供了更强的技术框架。基于面向对象的模型，是综合了面向对象和原型方法及重用技术的一种模型。该模型如图 4-5 所示。

该模型描述了软件从需求开始，通过检索重用构件库，一方面进行构件开发，另一方面进行需求开发，需求开发完成后，在进行面向对象分析过程中，它可以在重用构件库中读取构件，并快速建立 OOA（Object-Oriented Analysis ） 原型。同理，在进行面向对象设计时，它可以在重用构件库中读取构件，并快速建立 OOD（Object-Oriented Design）原型。最后利用生成技术，建造一个目标系统。在这个模型中，一个系统可以由重用构件组装而成，甚至通过组装可重用的子系统而创建更大的系统。

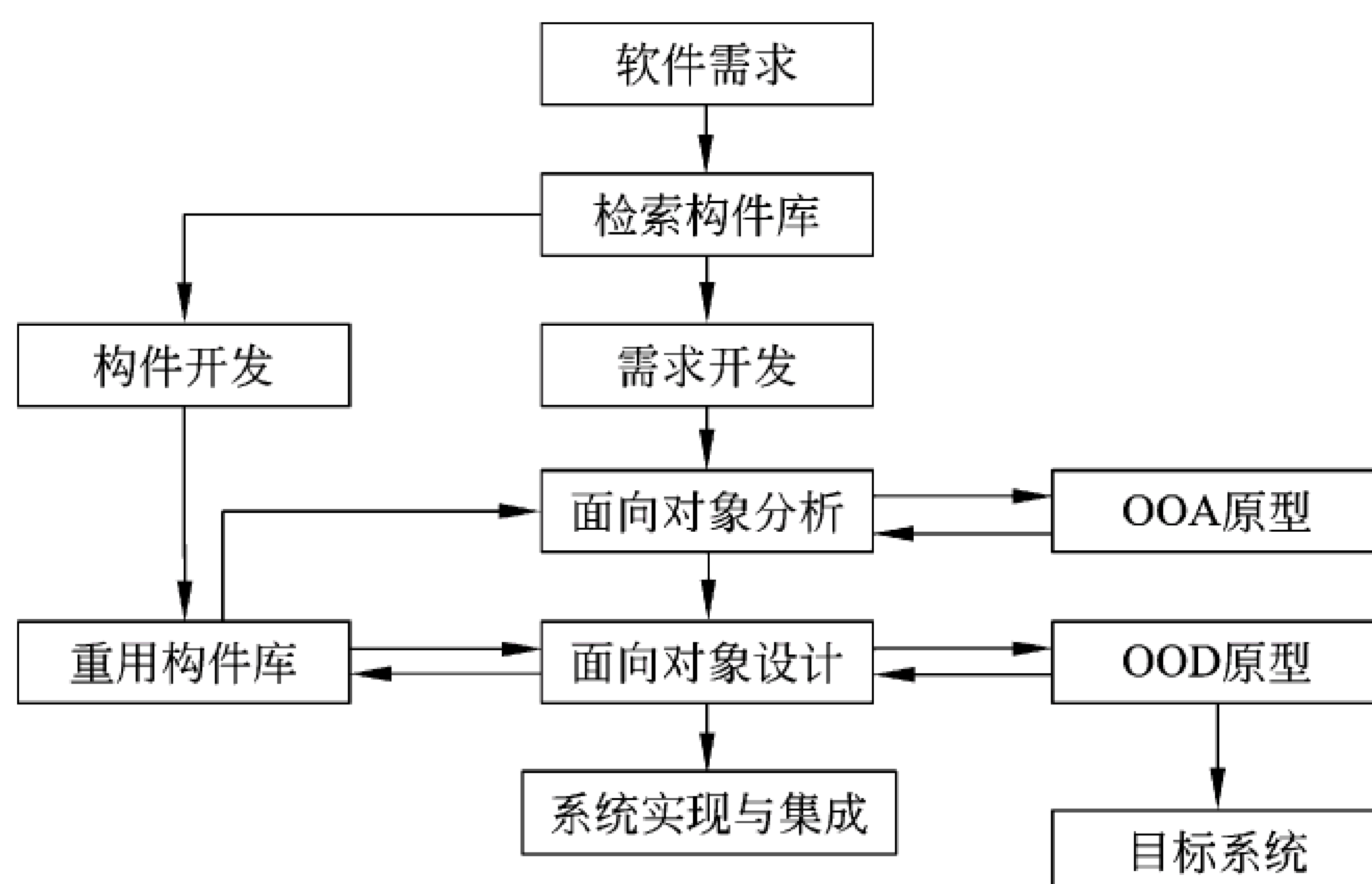


图 4-5　基于面向对象的模型

6．基于四代技术的模型

四代语言（4th Generation Language，4GL）是在大型数据库管理系统的基础上发展起来的程序设计语言。程序设计语言可分成机器语言、汇编语言、高级语言和第四代语

言，以及为人工智能领域应用而设计的语言——第五代语言。

4GL 目前还没有统一的定义，它的描述定义如下。

（1）用于快速开发应用软件的高产工具（重点强调了提高软件开发的生产率）。

（2）用于快速事务处理系统的高产工具（突出了主要应用领域）。

其主要特征描述如下。

（1）它是非过程化的语言，目的在于高效、直接地实现各种应用系统。它完全不用编程的方式来构造应用系统。程序员可以不再使用通常编程的方法、算法等来完成某一个功能，而是利用一些生成器，例如，菜单生成器、报表生成器、屏幕生成器、图形软件包等。在屏幕上以对话的交互方式，通过填表或操作屏幕上的窗口和按钮图标，或者在某一个按钮定义时加上适当的一段程序……从而构造用户需要的应用系统生成器自动生成源程序。

（2）它与数据库的关系密切，能够对大型数据库进行高效处理。它被广泛地应用于数据库管理系统中。

注意：第四代语言与通常的软件工程环境或计算机辅助软件工程（Computer-Aided Software Engineering，CASE）不同，CASE 支持软件开发的全过程，4GL 不支持应用软件开发的全过程，只侧重于支持应用软件开发过程中的设计阶段和实现阶段，特别是支持界面以及与界面有关的处理过程。

以 4GL 为核心的软件开发技术称为四代技术。使用四代技术可以给我们带来许多方便，在软件开发的时间、成本和质量等方面都会取得较好的效果，但它毕竟在系统开发全过程中应用有限。

4.1.3 敏捷方法

1．敏捷方法的特点

敏捷型方法主要有两个特点，这也是其区别于其他方法，尤其是重型方法的最主要的特征。

- 敏捷型方法是“适应性”（adaptive）而非“预设性”（predictive）的。重型方法试图对一个软件开发项目在很长的时间跨度内做出详细的计划，然后依计划进行开发。这类方法在计划制定完成后拒绝变化。而敏捷型方法则欢迎变化。其实，它的目的就是成为适应变化的过程，甚至能允许改变自身来适应变化。
- 敏捷型方法是，“面向人的”（people-oriented）而非“面向过程的”（process-oriented）。它们试图使软件开发工作能够利用人的特点，充分发挥人的创造能力。它们强调软件开发应当是一项愉快的活动。

下面是对上面两点的详细解释：

1）适应性和预设性

传统的软件开发方法的基本思路一般是从其他工程领域借鉴而来的，比如土木工程

等。在这类工程实践中，通常非常强调施工前的设计规划。只要图纸设计得合理并考虑充分，施工队伍可以完全遵照图纸顺利建造，并且可以很方便地把图纸划分为许多更小的部分交给不同的施工人员分别完成。

但是，软件开发与上面的土木工程有着显著的不同。软件的设计是难以实现的，并且需要昂贵的有创造性的人员。土木工程师在设计时所使用的模型是基于多年的工程实践的，而且一些设计上的关键部分都是建立于坚实的数学分析之上。而在软件设计中，完全没有类似的基础。我们对开发计划所能做的只是请专家审阅。这就使得我们无法将设计和实施分离开来，一些设计错误只能在编码和测试时才能发现。根本无法做出一个交给程序员就能直接编码的软件设计。

所以，软件过程不可能照搬其他工程领域原有的方法，需要有适应其特点的新的开发方法。

软件的设计之所以难以实现，问题在于软件需求的不稳定，从而导致软件过程的不可预测。但是传统的控制项目的模式都是针对可预测的环境的，在不可预测的环境下，就无法使用这些方法。

但是我们必须对这样的过程进行监控，以使得整个过程能向我们期望的目标前进。于是 Agile 方法引入“适应性”方法，该方法使用反馈机制对不可预测过程进行控制。

2）面向人而非面向过程

传统正规方法的目标之一是发展出这样一种过程，使得一个项目的参与人员成为可替代的部件。这样的一种过程将人看成是一种资源，他们具有不同的角色，如分析员、程序员、测试员及管理人员。个体是不重要的，只有角色才是重要的。这样考虑的一个重要的出发点就是：尽量减少人的因素对开发过程的影响。但是敏捷型方法则正好相反。

传统方法是让开发人员“服从”一个过程而非“接受”一个过程。但是一个常见的情况是：软件的开发过程是由管理人员决定的，而管理人员已经脱离实际开发活动相当长的时间了，如此设计出来的开发过程是难以为开发人员所接受的。

敏捷型过程还要求开发人员必须有权作技术方面的所有决定。IT 行业和其他行业不同，其技术变化速度非常之快。今天的新技术可能几年后就过时了。只有在第一线的开发人员才能真正掌握和理解开发过程中的技术细节。所以技术方面的决定必须由他们来做出。这样一来，就使得开发人员和管理人员在一个软件项目的领导方面有同等的地位，他们共同对整个开发过程负责。

敏捷方法特别强调开发中相关人员之间的信息交流。Alistair Cockburn 在对数十个项目的案例调查分析后得出一个结论，“项目失败的原因最终都可以追溯到信息没有及时准确地传递到应该接受它的人”。在开发过程中，项目的需求是在不断变化的，管理人员之间、开发人员之间以及管理人员和开发人员之间，都必须不断地了解这些变化，对这些变化做出反应，并实施在随后的开发过程中。敏捷方法还特别提倡直接的面对面交流。Alistair Cockburn 认为面对面交流的成本要远远低于文档交流的成本，因此，敏捷方法一

般都按照高内聚、松耦合的原则将项目划分为若干小组，以增加沟通，提高敏捷性及应变能力。

2．敏捷方法的核心思想

敏捷方法的核心思想主要有下面三点：

（1）敏捷方法是适应型，而非可预测型。与传统方法不同，敏捷方法拥抱变化，也可以说它的初衷就是适应变化的需求，利用变化来发展，甚至改变自己，最后完善自己。

（2）敏捷方法是以人为本，而非以过程为本。传统方法以过程为本，强调充分发挥人的特性，不去限制它。并且软件开发在无过程控制和过于严格繁琐的过程控制中取得一种平衡，以保证软件的质量。

（3）迭代增量式的开发过程。敏捷方法以原型开发思想为基础，采用迭代增量式开发，发行版本小型化。它根据客户需求的优先级和开发风险，制定版本发行计划，每一发行版都是在前一成功发行版的基础上进行功能需求扩充，最后满足客户的所有功能需求。

3．敏捷型方法的含义及其特征

我们把软件开发过程中拥有大量中间产品（如需求规约、设计模型等）和复杂控制的软件开发方法称为重型方法；由此，我们称中间产品较少的方法为轻型方法。从表象来看，重型方法注重开发文档的完备性和充分性；而敏捷型方法认为最根本的文档应该是源码，而不是繁琐的文档。从实质上说，有如下两方面更深层次的区别：

（1）敏捷型方法的思想是“自适应”的，而非如“预设”的重型方法试图预先固定需求并拟定详细开发计划；敏捷型方法适应需求的变化，甚至可以说其初衷就是针对变化的需求的。

（2）敏捷型方法的思考角度是“面向人”的，而非“面向开发过程”的。重型方法在实践原则中总是把开发者看作是一个泛化的生产要素，而忽视了作为决定性因素的人的特殊性；而敏捷型方法则强调以人为本，并贯穿实践始终。由上可知敏捷型方法其实是软件开发方法论从无到重型再进一步发展的成果。

4．敏捷方法的适用范围

实际上，满足工程设计标准的唯一文档是源代码清单。“软件项目的设计是一个抽象的概念。它涉及了程序的概括形状（shape）、结构以及每一模块、类和方法的详细形状。”系统设计得到了有关系统的一个清晰的“图像”，这一图像可以保持到首次发布。但随着项目的开发，程序“片段”就可能像不断腐化的“面包碎片”，发出“臭味”，并不断蔓延和积累，使得系统越来越难以维护，以至于不得不要求重新设计。但这样的重新设计是很难成功的。

因此，与这种传统的方法相比，敏捷方法比较适合需求变化比较大或者开发前期对需求不是很清晰的项目，以它的灵活性来适应需求的变化，有效地控制项目进度和成本。另外，敏捷方法对设计者、开发者和客户之间的有效沟通和及时反馈要求比较高，所以

不易在开发团队比较庞大的项目中实施，当然这也不是绝对的。

5. 敏捷方法的主要内容

敏捷方法的主要内容包括 4 个核心价值观和 12 条过程实践规则。4 个核心价值观分别为沟通、简单、反馈和勇气。沟通，它强调设计者、开发者和客户三者之间的有效交流是开发成功的关键；简单是设计和编码的指导原则，它强调只满足当前功能需求，不做假想设计，尽量使代码简单化；反馈，强调设计者、开发者和客户之间及时和详尽的意见反馈是开发成功的保证；勇气，是开发适应变化的前提，要求设计者和开发者在必需做出取舍或重构时，勇于抉择，勇于实践。

依据敏捷方法的 4 个核心价值观，提出 12 条过程实践规则，分别为简单设计、测试驱动、代码重构、结对编程、持续集成、现场客户、发行版本小型化、系统隐喻、代码集体所有制、规划策略、规范代码、40 小时工作机制。

6. 主要敏捷方法简介

手工作坊式的软件生产方式已经被无数次的项目失败证明为低效和应被舍弃的：传统软件开发方法（如 ISO9000 和 CMM）在规范和保证开发进程的同时，由于其繁琐的过程控制和严格的文档要求招致了开发者潜在的抵触；此外，开发人员流动性大于软件的可持续开发之间的矛盾日渐显露，如何保证软件的高可传承性以及尽可能地延长软件生命周期成了摆在开发者和管理者面前的难题。为了应对这种局面，近年来，已经出现很多敏捷型方法，它们有许多的共同特征，但也有一些重要的不同之处。这里就其中影响比较大的几种敏捷方法作一些简单的介绍。

（1）XP（Extreme Programming，极限编程）在所有的敏捷型方法中，XP 是最引人瞩目的。它源于 Smalltalk 圈子，特别是 Kent Beck 和 Ward Cunningham 在 20 世纪 80 年代末的密切合作。XP 在一些对费用控制严格的公司中的使用，已经被证明是非常有效的。

（2）Cockburn 的水晶系列方法。水晶系列方法是由 Alistair Cockburn 提出的。它与 XP 方法一样，都有以人为中心的理念，但在实践上有所不同。Alistair 考虑到人们一般很难严格遵循一个纪律约束很强的过程，因此，与 XP 的高度纪律性不同，Alistair 探索了用最少纪律约束而仍能成功的方法，从而在产出效率与易于运作上达到一种平衡。也就是说，虽然水晶系列不如 XP 那样的产出效率，但会有更多的人能够接受并遵循它。

（3）开放式源码。这里提到的开放式源码指的是开放源码界所用的一种运作方式。开放式源码项目有一个特别之处，就是程序开发人员在地域上分布很广，这使得它和其他敏捷方法不同，因为一般的敏捷方法都强调项目组成员在同一地点工作。开放源码的一个突出特点就是查错排障（debug）的高度并行性，任何人发现了错误都可将改正源码的“补丁”文件发给维护者。然后由维护者将这些“补丁”或是新增的代码并入源码库。

（4）SCRUM。SCRUM 已经出现很久了，像前面所论及的方法一样，该方法强调这样一个事实，即明确定义了的可重复的方法过程只限于在明确定义了的可重复的环境中，

为明确定义了的可重复的人员所用，去解决明确定义了的可重复的问题。

（5）Coad 的功用驱动开发方法（Feature Driven Development，FDD）。FDD 是由 Jeff De Luca 和大师 Peter Coad 提出来的。像其他方法一样，它致力于短时的迭代阶段和可见可用的功能。在 FDD 中，一个迭代周期一般是两周。

在 FDD 中，编程开发人员分成两类：首席程序员和“类”程序员（class owner）。首席程序员是最富有经验的开发人员，他们是项目的协调者、设计者和指导者，而“类”程序员则主要做源码编写。

（6）ASD 方法。ASD （Adaptive Software Development）方法由 Jim Highsmith 提出，其核心是三个非线性的、重叠的开发阶段：猜测、合作与学习。

4.1.4 RUP

1. RUP 概述

RUP（Rational Unified Process）根据字面理解，可以知道 RUP 包括三方面的意思，即 Rational、Unified 和 Process。Rational 表示 RUP 是由 Rational 公司提出的，Unified 表示 RUP 是最佳开发经验总结，而 Process 表示 RUP 是一个软件开发过程。

2. RUP 的生命周期

RUP 软件开发生命周期是一个二维的软件开发模型，RUP 中有 9 个核心工作流，这 9 个核心工作流如下。

- 业务建模（business modeling）：理解待开发系统所在的机构及其商业运作，确保所有参与人员对待开发系统所在的机构有共同的认识，评估待开发系统对所在机构的影响。
- 需求（requirements）：定义系统功能及用户界面，使客户知道系统的功能，使开发人员理解系统的需求，为项目预算及计划提供基础。
- 分析与设计（analysis & design）：把需求分析的结果转化为分析与设计模型。
- 实现（implementation）：把设计模型转换为实现结果，对开发的代码做单元测试，将不同实现人员开发的模块集成为可执行系统。
- 测试（test）：检查各子系统的交互与集成，验证所有需求是否均被正确实现，对发现的软件质量上的缺陷进行归档，对软件质量提出改进建议。
- 部署（deployment）：打包、分发、安装软件，升级旧系统；培训用户及销售人员，并提供技术支持。
- 配置与变更管理（configuration & change Management）：跟踪并维护系统开发过程中产生的所有制品的完整性和一致性。
- 项目管理（project management）：为软件开发项目提供计划、人员分配、执行、监控等方面的指导，为风险管理提供框架。
- 环境（environment）：为软件开发机构提供软件开发环境，即提供过程管理和工具的支持。

需要说明的是表示核心工作流的术语 discipline，在 RUP 2000 以前用的是 core workflow 这个术语，但在最新的版本中已改为用 discipline。discipline 的中文意义较多，根据 RUP 的定义，discipline 是相关活动的集合，这些活动都和项目的某一个方面有关，如这些活动都是和业务建模相关的，或者都是和需求相关的，或者都是和分析设计相关的等等。

RUP 把软件开发生命周期划分为多个循环（cycle），每个 cycle 生成产品的一个新的版本，每个 cycle 依次由 4 个连续的阶段（phase）组成，每个阶段完成确定的任务。这 4 个阶段如下。

- 初始（inception）阶段：定义最终产品视图和业务模型，并确定系统范围。
- 细化（elaboration）阶段：设计及确定系统的体系结构，制定工作计划及资源要求。
- 构造（construction）阶段：构造产品并继续演进需求、体系结构、计划直至产品提交。
- 移交（transition）阶段：把产品提交给用户使用。

每一个阶段都由一个或多个连续的迭代（iteration）组成。迭代并不是重复地做相同的事，而是针对不同用例的细化和实现。每一个迭代都是一个完整的开发过程，它需要项目经理根据当前迭代所处的阶段以及上次迭代的结果，适当地对核心工作流中的行为进行裁剪。

在每个阶段结束前有一个里程碑（milestone）评估该阶段的工作。如果未能通过该里程碑的评估，则决策者应该做出决定，是取消该项目还是继续做该阶段的工作。

3．RUP 中的核心概念

RUP 中定义了如下一些核心概念，理解这些概念对于理解 RUP 很有帮助。

- 角色（Role）——who 的问题：角色描述某个人或一个小组的行为与职责。RUP 预先定义了很多角色，例如体系结构师（architect）、设计人员（designer）、实现人员（implementer）、测试员（tester）和配置管理人员（configuration manager）等，并对每一个角色的工作和职责都做了详尽的说明。
- 活动（activity）——how 的问题：活动是一个有明确目的的独立工作单元。
- 制品（artifact）——what 的问题：制品是活动生成、创建或修改的一段信息。也有些书把 artifact 翻译为产品、工件等，和制品的意思差不多。
- 工作流（workflow）——when 的问题：工作流描述了一个有意义的连续的活动序列，每个工作流产生一些有价值的产品，并显示了角色之间的关系。

RUP 2003 对这些概念有比较详细的解释，并用类图描述了这些概念之间的关系，除了 role、activity、artifact 和 workflow 这 4 个核心概念外，还有其他一些基本概念，如工具教程（tool mentor）、检查点（checkpoints）、模板（template）和报告（report）等。

4．RUP 的特点

与别的软件开发过程相比，RUP 具有自己的特点，即 RUP 是用例驱动的、以体系结构为中心的、迭代和增量的软件开发过程。下面对这些特点做进一步的分析。

1）用例驱动

RUP 中的开发活动是用例驱动的，即需求分析、设计、实现和测试等活动都是用例驱动的。

2）以体系结构为中心

RUP 中的开发活动是围绕体系结构展开的。对于软件体系结构，目前还没有一个统一的精确的定义，不同的人对软件体系结构有不同的认识。Mary Shaw 和 David Garlan 对软件体系结构的定义是：软件体系结构是关于构成系统的元素、这些元素之间的交互、元素和元素之间的组成模式以及作用在这些组成模式上的约束等方面的描述。具体来说，软件体系结构刻画了系统的整体设计，它去掉了细节部分，突出了系统的重要特征。

软件体系结构的设计和代码设计无关，也不依赖于具体的程序设计语言。软件体系结构是软件设计过程中的一个层次，这一层次超越计算过程中的算法设计和数据结构设计。体系结构层次的设计问题包括系统的总体组织和全局控制、通讯协议、同步、数据存取、给设计元素分配特定功能、设计元素的组织、物理分布、系统的伸缩性和性能等。

体系结构的设计需要考虑多方面的问题：在功能性特征方面要考虑系统的功能；在非功能性特征方面要考虑系统的性能、安全性和可用性等；与软件开发有关的特征要考虑可修改性、可移植性、可重用性、可集成性和可测试性等；与开发经济学有关的特征要考虑开发时间、费用、系统的生命期等。当然，这些特征之间有些是相互冲突的，一个系统不可能在所有的特征上都达到最优，这时就需要系统体系结构设计师在各种可能的选择之间进行权衡。

对于一个软件系统，不同人员所关心的内容是不一样的，因此软件的体系结构是一个多维的结构，也就是说，会采用多个视图（view）来描述软件体系结构。打个比喻，对于一座大厦，会有大厦的电线布线结构、电梯布局结构、水管布局结构等，对于大厦的建设和维护人员来说，有些人关心大厦的电线布局，有些人关心大厦的电梯布局，还有些人关心水管布局，对于不同类型的人员，只需要提供这类人员关心的视图即可（一个视图可以用一个或多个图来描述），所有这些视图组成了大厦的体系结构。至于采用多少个视图，采用什么视图较好，不同的人就有不同的观点了。

在 RUP 中，是采用如图 4-6 所示的“4+1”视图模型来描述软件系统的体系结构。

在“4 +1”视图模型中，分析人员和测试人员关心的是系统的行为，因此会侧重于用例视图；最终用户关心的是系统的功能，因此会侧重于逻辑视图；程序员关心的是系统的配置、装配等问题，因此会侧重于实现视图；系统集成人员关心的是系统的性能、可伸缩性、吞吐率等问题，因此会侧重于进程视图；系统工程师关心的是系统的发布、

安装、拓扑结构等问题，因此会侧重于部署视图。

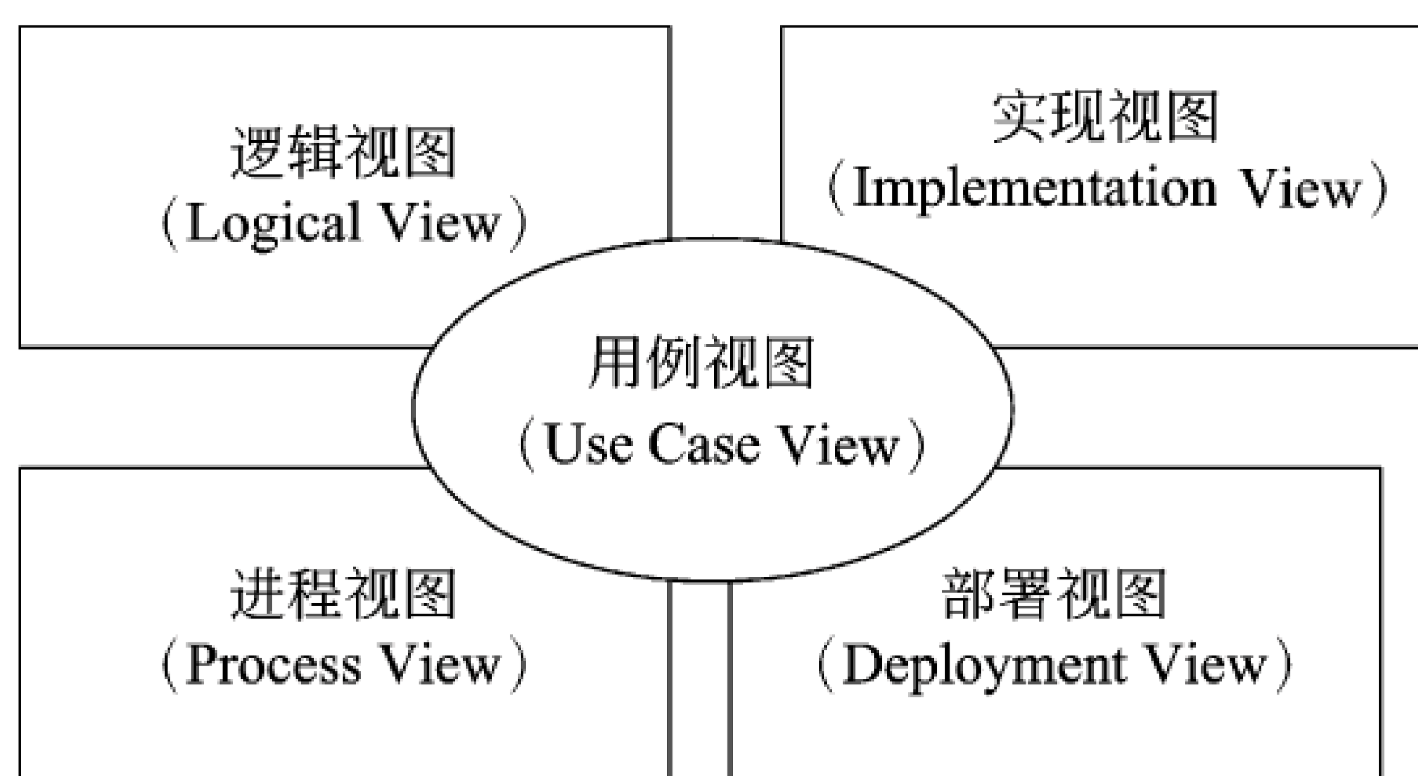


图 4-6 “4 +1”视图模型

3）迭代与增量

RUP 强调要采用迭代和增量的方式来开发软件，把整个项目开发分为多个迭代过程。在每次迭代中，只考虑系统的一部分需求，进行分析、设计、实现、测试和部署等过程，每次迭代是在已完成部分的基础上进行的，每次增加一些新的功能实现，以此进行下去，直至最后项目的完成。软件开发采用迭代和增量的方式有以下好处：

（1）在软件开发的早期就可以对关键的、影响大的风险进行处理。

（2）可以提出一个软件体系结构来指导开发。

（3）可以更好地处理不可避免的需求变更。

（4）可以较早地得到一个可运行的系统，鼓舞开发团队的士气，增强项目成功的信心。

（5）为开发人员提供一个能更有效工作的开发过程。

5．RUP 裁剪

RUP 是一个通用的过程模板，包含了很多关于开发指南、开发过程中产生的制品、开发过程中所涉及的各种角色的说明。RUP 可用于各种不同类型的软件系统、不同的应用领域、不同类型的开发机构、不同功能级别、不同规模的项目中。RUP 非常庞大，没有一个项目会使用 RUP 中的所有东西，针对具体的开发机构和项目，应用 RUP 时还要做裁剪，也就是要对 RUP 进行配置。RUP 就像是一个元过程（meta-process），通过对 RUP 进行裁剪可以得到很多不同的软件开发过程，这些软件开发过程可以看作 RUP 的具体实例，这些具体的开发过程实例适合于不同的开发机构和项目的需要。

针对一个软件项目，RUP 裁剪可分为以下几步：

（1）确定本项目的软件开发过程需要哪些工作流。RUP 的 9 个核心工作流并不总是需要的，可以根据项目的规模、类型等对核心工作流做一些取舍。如嵌入式软件系统项目一般就不需要业务建模这个工作流。

（2）确定每个工作流要产出哪些制品。如规定某个工作流应产出哪些类型的文档。

（3）确定 4 个阶段之间（初始阶段、细化阶段、构造阶段和移交阶段）如何演进。确定阶段间演进要以风险控制为原则，决定每个阶段要执行哪些工作流，每个工作流执行到什么程度，产出的制品有哪些，每个制品完成到什么程度等。

（4）确定每个阶段内的迭代计划。规划 RUP 的 4 个阶段中每次迭代开发的内容有哪些。迭代是 RUP 非常强调的一个概念，可以进一步降低开发风险。

（5）规划工作流内部结构。工作流不是活动的简单堆积，工作流涉及角色、活动和制品，工作流的复杂程度与项目规模及角色多少等有很大关系，这一步决定裁剪后的 RUP 要设立哪些角色。最后，规划工作流的内部结构，通常用活动图的形式给出。

在上面的 5 个步骤中，第（5）步是对 RUP 进行裁剪的难点。如果从“软件开发过程也是软件”的角度来看，对 RUP 进行裁剪可以看作是软件过程开发的再工程。

4.1.5 软件系统工具

软件系统工具的种类繁多，很难有统一的分类方法。通常可以按软件过程活动将软件工具分为软件开发工具、软件维护工具 、软件管理和软件支持工具。

1．软件开发工具

对应软件开发过程的各种活动，软件开发工具有需求分析工具、设计工具、编码与排错工具、测试工具等。

1）需求分析工具

需求分析工具用以辅助软件需求分析活动，辅助系统分析员从需求定义出发，生成完整的、清晰的、一致的功能规范。功能规范是软件所要完成的功能精确而完整的陈述，描述该软件要做什么及只做什么，是软件开发者和用户间的契约，同时也是软件设计者的和实现者的依据。功能规范应正确、完整地反映用户对软件的功能要求，其表达是清晰的、无歧义的。需求分析工具的目标就是帮助分析员形成这样的功能规范。

按描述需求定义的方法可将需求分析工具分为基于自然语言或图形描述的工具和基于形式化需求定义语言的工具。

2）基于自然语言或图形描述的工具

这类工具采用分解与抽象等基本手段，对用户问题逐步求精，并在检测机制的辅助下，发现其中可能存在的问题（如一致性），通过对问题描述的修改，逐步形成能正确反映用户需求的功能规范。它能帮助分析员提高需求文档的质量，降低功能规范的维护费用。这里以支持结构化方法的需求分析工具为例介绍。

结构化分析方法采用数据流图的描述方法，分析的主要结果是一套分层的数据流图和一个数据词典。结构化需求分析工具通常由图形编辑器、数据词典管理器和检测机制三部分组成。使用图形编辑器绘制数据流图，该图形编辑器应支持图形的分层结构，以构成分层数据流图。在构造数据流图的同时把数据流图的有关信息（如加工名、数据流名、数据项、文件名等及它们之间的联系）填入数据词典。在填写数据词典的过程中，

数据词典管理器即可查出重名等错误。在构造出分层数据流图后，可通过检测机制来检查分层数据流图的合法性，可发现诸如父图与子图不平衡，遗漏的数据流（如无输入数据流或无输出数据流的加工），只有读文件没有写文件或只有写文件没有读文件等错误。然后将修改后的数据流图和词典与用户交流，考察它是否符合用户的功能需求。若不一致，再使用图形编辑器进行修改。需求分析工具还应具备同步修改的功能，即修改数据流图的同时也修改数据词典中的有关信息，以保持数据流图与数据词典的一致性。经过多次反复的交流和修改，使功能规范逐步达到准确、完整和一致，最后形成有效的功能规范。除此以外，该工具还可浏览数据词典，生成各种统计或查询报告。

3）基于形式化需求定义语言的工具

基于形式化需求定义语言的工具大多以基于知识的需求智能助手的形式出现，并把人工智能技术运用于软件工程。这类工具通常具有一个知识库和一个推理机制。知识库中存放需求分析所需的公共知识，以及特定的应用领域知识。这些知识能用来理解需求定义中的省略写法，能部分消除不完整性和歧义性。推理机制能容忍需求定义的无序性，部分解决描述中的不一致性。这类工具接受用形式化语言书写的功能描述，运用知识库中的知识，通过推理，发现需求定义中的矛盾和不足，经补充、更新知识库中的知识和规则，以及与系统分析员的不断交互，得到完整的功能规范。

4）其他需求分析工具

可执行规范语言以及原型技术为需求分析工具提供了另一条实现途径，这些工具通过运行可执行规范或原型，将有关的结果显示给用户和系统分析员，以便进行需求确认。

2．设计工具

设计工具用以辅助软件设计活动，辅助设计人员从软件功能规范出发，得到相应的设计规范。

设计规范是符合功能规范和需求定义中所指定的功能及性能要求，对软件的组织或其组成部分的内部结构的描述。通常设计规范分成概要设计规范和详细设计规范。概要设计规范描述软件的功能模块及其相互关系，说明模块的处理过程和外部行为，同时还应描述数据的逻辑结构。详细设计规范描述每个模块的处理算法及涉及到的全部数据结构。设计规范是程序员进行编程活动的依据。

3．编码与排错工具

编码工具和排错工具用以辅助程序员进行编码活动。编码工具辅助程序员用某种程序语言编制源程序，并对源程序进行翻译，最终转换成可执行的代码，因此编码工具通常与编码所使用的程序语言密切相关。排错工具用来辅助程序员寻找源程序中错误的性质和原因，并确定其出错的位置。由于源程序一般以正文的形式出现，必须有编辑器将它输入，并进行浏览、编辑和修改。又由于源程序的编写往往不会一次成功，需要不断寻找其中的错误并加以纠正。编码工具和排错工具是编程活动的重要辅助工具，也是最早出现的软件工具。

1）编码工具

主要有编辑程序、汇编程序、编译程序和生成程序等。

- 编辑程序：编辑程序用以输入源程序，并对其进行增加、删除和修改等操作。除常见的以字符为单位进行编辑的正文编辑程序外，还有面向程序语言语法单位的语法制导编辑程序和混合编辑程序。语法制导编辑程序也称结构化编辑程序，它可根据程序语言的语法规则提供编辑时的语法制导和检查，可以一次扩展或删除一个语法单位，如语句、表达式等，从而确保输入的源程序在语法上是正确的。混合编辑程序兼有正文编辑和语法制导编辑两种方法。
- 汇编程序：汇编程序用以将汇编语言书写的程序翻译成等价的机器语言程序。如果汇编程序所生成的机器指令代码是另一种计算机的机器指令，便称这类汇编程序为交叉汇编程序。
- 编译程序：编译程序用以将高级程序语言书写的程序翻译成等价的低级程序语言程序。
- 生成程序：生成程序通常根据与领域有关的甚高级语言或某种专用语言描述的用户需求，自动生成高级程序语言或低级程序语言描述的程序。例如，词法分析生成程序 LEX，它根据正规表达式表示的词法规则，自动生成词法分析程序的代码段，来实现能识别所说明的正规表达式的有限状态自动机。

一般把汇编程序和编译程序看作语言处理程序，生成程序属于软件自动化范围，所以编码工具主要是指编辑程序。

2）排错工具

已有的排错工具主要有源代码排错程序和排错程序生成程序两类。

- 源代码排错程序：源代码排错程序用以帮助程序员理解程序的执行状态，可通过对程序执行过程中各种状态的判别进行程序错误的识别、定位及改正。
- 排错程序生成程序：排错程序生成程序是一种通用的排错工具。对给定的程序语言，它能生成一个相应的源代码排错程序。

4．软件维护工具

软件维护工具辅助软件维护过程中的活动，辅助维护人员对软件代码及其文档进行各种维护活动。软件维护工具主要有版本控制工具、文档分析工具、开发信息库工具、逆向工程工具和再工程工具等。

1）版本控制工具

在软件开发和维护过程中一个软件会有多个版本，版本控制工具用来存储、更新、恢复和管理一个软件的多个版本。UNIX 的（Source Code Control System，SCCS）（源代码控制系统）是版本控制工具的典型代表。SCCS 能为正文文件的多个版本建立一棵版本树，第一版完整储存文本全文，以后各版只存放它之前版本的不同之处，在任何时刻 SCCS 只允许对一个当前版本进行修改和提交。通过版本树维护版本的更新历史，并允

许恢复到以前的某个版本。

2）文档分析工具

文档分析工具用以对软件开发过程中形成的文档进行分析，给出软件维护活动所需的维护信息。例如，基于数据流图的需求文档分析工具可给出对数据流图的某个成分（如加工）进行维护时的影响范围及被影响范围，以便在该成分修改的同时考虑其影响范围内的其他成分是否也要修改。基于模块结构图的设计文档分析工具可以给出对模块变量进行维护时的影响及被影响范围。针对程序文档的源代码分析工具可给出模块、全局变量、局部变量的定义、引用情况，它还可以进行程序分片。程序分片是把程序中与指定的数据项或数据结构有关的程序代码抽出来，过滤掉与其无关的代码，以便维护人员高效地理解和把握他所关心的部分。文档分析工具还可得到被分析的文档的有关信息，如文档各种成分的个数、定义及引用情况等。

3）开发信息库工具

开发信息库工具用来维护软件项目的开发信息，包括对象、模块等。它记录每个对象的修改信息（已确定的错误及重要改动）和其他变形（如抽象数据结构的多种实现）；维护对象和与之有关信息之间的关系，包括模块的设计者、新版本中模块的改动及其与错误、测试用例、测试结果之间的联系等；其他必须记录的信息，包括用来生成此软件产品的所有工具的版本信息（如编译程序、连接程序、生成程序的版本号），所采用的命令语言程序和系统库以及测试用例版本和测试报告。

4）逆向工程工具

在软件生存周期中，将某种形式表示的软件转换成更高抽象形式表示的软件的活动称为逆向工程。例如，用反汇编工具将机器语言代码转换成汇编语言代码，用反编译工具将汇编语言代码或机器语言代码转换成某种高级程序语言源程序，之后再将源程序转换成详细设计的某种表示形式，这都属于逆向工程的范畴。逆向工程工具就是辅助软件人员进行这种逆向工程活动的软件工具。若软件缺乏必要的文档，原先的开发人员又已调离，就需使用逆向工程工具来理解原有的软件。

5）再工程工具

再工程工具用来支持重构一个功能和性能更为完善的软件系统。目前的再工程工具主要集中在代码重构、程序结构重构和数据结构重构等方面。

代码重构工具可把用一种程序语言书写的程序转换成用另一种程序语言书写的或适用于不同硬件平台的程序，例如 FORTRAN 到 C 的转换工具。程序结构重构工具接受一个非结构化或结构化程度较低的源程序，构造出行为等价的结构化程序。数据结构重构工具通过对数据描述的分析，重构新的数据结构。

5．软件管理和软件支持工具

软件管理过程和软件支持过程往往要涉及到软件生存周期中的多个活动，软件管理和软件支持工具用来辅助管理人员和软件支持人员的管理活动和支持活动，以确保软件

高质高效地完成。

辅助软件管理和软件支持的工具有很多，其中常用的工具有项目管理工具、配置管理工具、软件评价工具等。

1）项目管理工具

项目管理工具用来辅助软件的项目管理活动。通常项目管理活动包括项目的计划、调度、通信、成本估算、资源分配及质量控制等。一个项目管理工具通常把重点放在某一个或某几个特定的管理环节上，而不提供对管理活动包罗万象的支持。

例如成本估算工具，采用某种成本估算模型（如 COCOMO 模型）对项目的成本进行估算。它可以通过间接的测量（如对代码行和功能点的测量）来估算项目的规模大小，并描述总的项目特征，如问题的复杂度、开发组经验和过程成熟度等。然后按一定的估算模型估算出项目的工作量、工期和开发人数等。当项目截止期限变更时，可检测它对整个开发成本的影响。

2）配置管理工具

配置管理工具用以辅助完成软件配置项的标识、版本控制、变化控制、审计和状态统计等基本任务，使各配置项的存取、修改和系统生成易于实现，从而简化审计过程，改进状态统计，减少错误，提高系统的质量。

3）软件评价工具

软件评价工具用以辅助管理人员进行软件质量保证的有关活动。它通常可按某个软件质量模型（如 McCall 软件质量模型，ISO 软件质量度量模型等）对被评价的软件进行度量，然后得到相关的软件评价报告。目前许多度量指标还不能定量化，需要通过专家评分，再将得分送给软件评价工具。对一些已经定量化的度量指标则可利用评价工具自动获取。有的评价工具还可分析被评价程序的程序结构，根据某种软件复杂性模型（如 Mc-Cabe 的环路复杂度等）对被评价的程序进行复杂性度量。软件评价工具有助于软件的质量控制，对确保软件的质量有重要的作用。

4）软件开发工具的评价和选择

现在各类软件开发工具十分丰富，有免费的，有价格便宜的，也有昂贵的。评价和选择适合本人、本单位、本项目的软件开发工具，可以根据以下标准来衡量软件开发工具的优劣。

（1）功能。软件开发工具不仅要实现所遵循的功能需求，支持用户所选定的开发方法，还应能检查与之相关的方法学能否正确执行，并保证产生与方法学一致的输出结果。

（2）易用性。软件开发工具应有十分友好的用户界面，用户乐于使用；工具应能剪裁和定制，以适应特定用户的需要；工具应能提示用户的交互操作，提供简单有效的执行方式；工具还应能检查用户的操作错误，尽可能自动改正错误。

（3）稳健性。一个好的软件开发工具应能长期可靠地使用，并能适应环境或其他条件变化的要求；即使在非法操作或故障情况下，也不应导致严重后果。

（4）硬件要求和性能。软件开发工具的性能（如响应速度、占用存储空间的大小等），将直接影响工具的使用效果。合理的性能和对硬件的要求可以使机器的资源能被有效地加以利用，使用户的投资发挥最大的作用。

（5）服务和支持。软件开发工具的生产厂商应能为该工具提供有效的技术服务（如培训、咨询、版本更新等），工具的文档应该齐全、通俗易懂。

4.2　需求管理

软件需求开发的最终文档经过评审批准后，则定义了开发工作的需求基线（baseline）。这个基线在客户和开发者之间构筑了计划产品功能需求和非功能需求的一个约定（agreement）。需求约定是需求开发和需求管理之间的桥梁。

需求管理是一个对系统需求变更、了解和控制的过程。需求管理过程与需求开发过程相互关联，当初始需求导出的同时就启动了需求管理规划，一旦形成了需求文档的初稿，需求管理活动就开始了。需求管理的主要活动如图 4-7 所示。

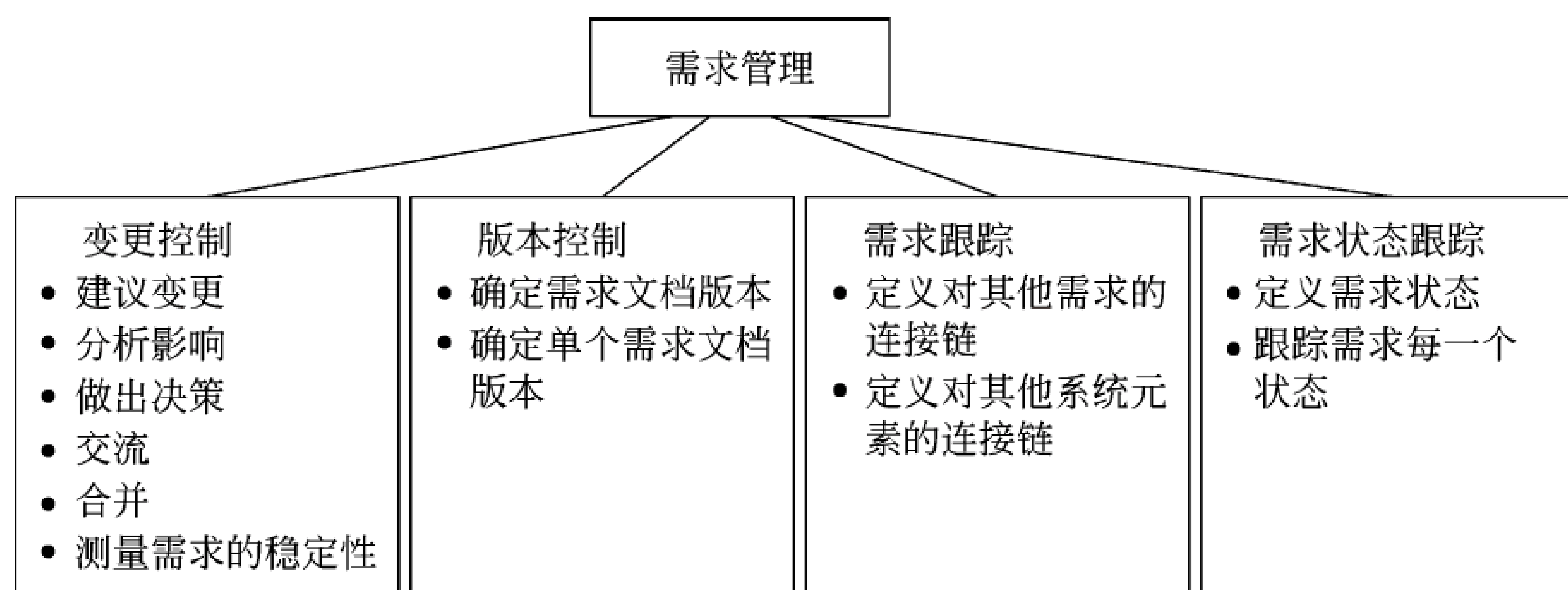


图 4-7　需求管理的主要活动

需求管理强调：

（1）控制对需求基线的变动。

（2）保持项目计划与需求一致。

（3）控制单个需求和需求文档的版本情况。

（4）管理需求和联系链，或者管理单个需求和其他项目可交付产品之间的依赖关系。

（5）跟踪基线中的需求状态。

4.2.1　需求管理原则

过程能力成熟度模型（Capability Maturity Model，CMM）在软件开发机构中被广泛用来指导软件过程改进。该模型描述了软件处理能力的 5 个成熟级别。为了达到过程能

力成熟度模型的第二级，组织机构必须具有 6 个关键过程域（Key Process Areas，KPA）。

需求管理是其中之一，它的目标如下。

（1）为软件需求建立一个基线，提供给软件工程和管理使用。

（2）软件计划、产品和活动与软件需求保持一致。

关于需求管理过程域内的原则和策略，可以参考：

（1）需求管理的关键过程领域不涉及收集和分析项目需求，而是假定已收集了软件需求，或者已由更高一级的系统给定了需求。一旦需求获得并且文档化了，软件开发组和有关的团队（例如质量保证和测试组）需要评审文档。发现问题应与客户或者其他需求源协商解决。软件开发计划是基于已确认的需求。

（2）开发人员在向客户以及有关部门承诺（commitment）某些需求之前，应该确认需求和约束条件、风险、偶然因素、假定条件等。也许不得不面对由于技术因素或者进度等原因，承诺一些不现实的需求。但是，决不要承诺任何无法实现的事。

（3）关键处理领域同样建议通过版本控制和变更控制来管理需求文档。版本控制确保随时能知道在项目开发和计划中正在使用的需求的版本情况。变更控制提供了支配下的规范的方式来统一需求变更，并且基于业务和技术的因素来同意或者反对建议的变更。当在开发中修改、增加、减少需求时，软件开发计划应该随时更新，确保与新的需求保持一致。

4.2.2 需求规格说明的版本控制

在软件开发过程中，可能出现测试人员使用已过时的软件规格说明，结果发现了一大堆的错误，为了避免这种情况的发生，需求规格说明的版本管理就显得非常重要了。

版本控制是管理需求的一个必要方面，必须统一确定需求文档的每一个版本。软件开发组的每一个成员必须得到需求的当前版本。当需求发生变更时，应该清楚地把变更写成文档，并且及时通知所有涉及的人员。为了尽量减少困惑、冲突和误传，应该仅允许指定的人员来更新需求。

每一个新版本的需求文档，应该公布其包括修正版本在内的历史情况，例如，变更的内容、变更日期、变更人员的姓名以及变更的原因等。任何新文档的第一版应当标记为“1.0 版（草案 1）”，下一稿标记为“1.0 版（草案 2）”，在文档被确认为基线前，草案数可以随着改进逐次增加，当文档被确认为基线后，被标记为“1.0 正式版”。以后，如果有较小的修改，可以标记为“1.1 版（草案 1）”，如果有较大的修改，可以标记为“2.0 版（草案 1）”。这种方法可以很清楚地区分草稿和文档定稿的版本。

4.2.3 需求属性

除了文本，每一个功能需求应该有一些相关的信息与它联系，我们把这些信息称为需求属性。对于一个大型的复杂项目来说，丰富的属性类别显得尤为重要。例如，在文

档中考虑和明确如下属性：

- 创建需求的时间。
- 需求的版本号。
- 创建需求的作者。
- 负责认可该软件需求的人员。
- 需求状态。
- 需求的原因和根据。
- 需求涉及的子系统。
- 需求涉及的产品版本号。
- 使用的验证方法或者接受的测试标准。
- 产品的优先级或者重要程度。
- 需求的稳定性。

4.2.4　需求变更

一个大型软件系统的需求总是有变化的，原因是该系统通常是要解决一些复杂和难度大的问题，而一些问题不可能一次就被完全定义；此外，开发者对问题的理解可能在变化，这些变化也反映到需求中。

对许多项目来说，系统软件总需要不断完善，一些需求的改进是合理的而且不可避免，要使得软件需求完全不变更，也许是不可能的，但毫无控制的变更是项目陷入混乱、不能按进度完成，或者软件质量无法保证的主要原因之一。事实上，迟到的需求变更会对已进行的工作产生非常大的影响。如果不控制范围的扩展，将使我们持续不断地采纳新功能，而且要不断地调整资源、进度或者质量标准，是极为有害的。如果每一个建议的需求变更都采用，该项目将有可能永远不能完成。

软件需求文档应该精确描述要交付的产品，这是一个基本的原则。为了使开发组织能够严格控制软件项目，应该确保以下事项：

- 仔细评估已建议的变更。
- 挑选合适的人选对变更做出决定。
- 变更应及时通知所有涉及的人员。
- 项目要按一定的程序来采纳需求变更。

1．变更控制过程

一个好的变更控制过程，给项目风险承担者提供了正式的建议需求变更机制。通过这些处理过程，项目负责人可以在信息充分的条件下做出决策。我们可以通过变更控制过程来跟踪已建议变更的状态，使已建议的变更确保不会丢失或疏忽。一旦确定了需求基线，应该使所有已建议的变更都遵循变更控制过程。

需求变更管理过程如图 4-8 所示。

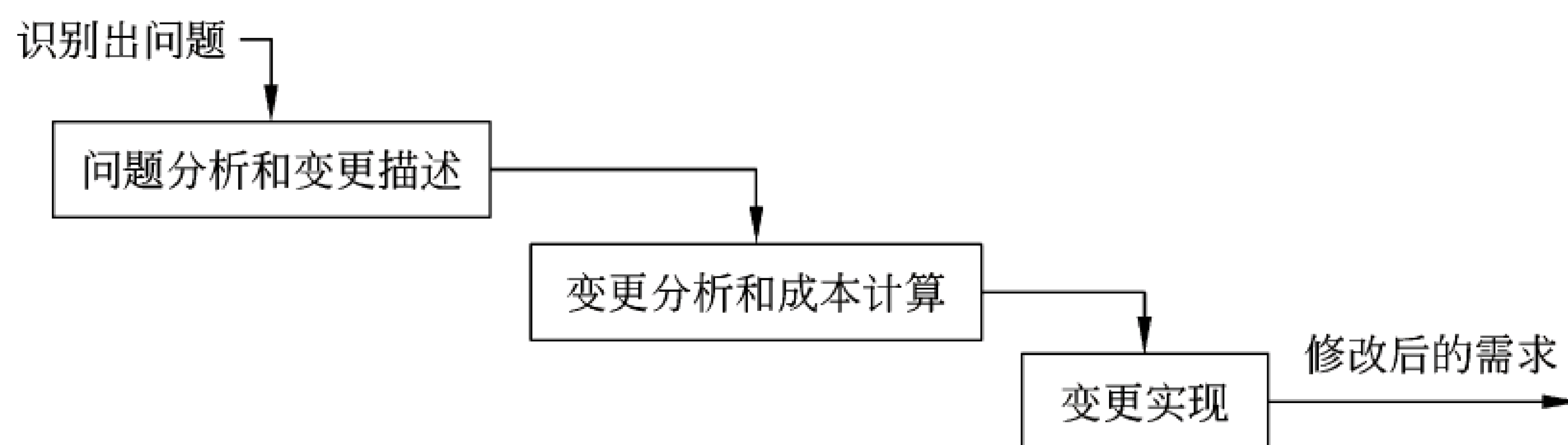


图 4-8 需求变更管理过程

（1）问题分析和变更描述。这是识别和分析需求问题或者一份明确的变更提议，以检查它的有效性，从而产生一个更明确的需求变更提议。

（2）变更分析和成本计算。使用可追溯性信息和系统需求的一般知识，对需求变更提议进行影响分析和评估。变更成本计算应该包括对需求文档的修改、系统修改的设计和实现的成本。一旦分析完成并且被确认，应该进行是否执行这一变更的决策。

（3）变更实现。这要求需求文档和系统设计以及实现都要同时修改。如果先对系统的程序做变更，然后再修改需求文档，这几乎不可避免地会出现需求文档和程序的不一致。

变更控制过程并不是给变更设置障碍。相反地，它是一个渠道和过滤器，通过它可以确保采纳最合适的变更，使变更产生的负面影响降到最低。变更过程应该做成文档，而且要简单、有效。

控制需求变更与项目的其他配置管理决策也有着密切的联系。项目管理应该达成一个策略，用来描述如何处理需求变更，而且策略具有现实可行性。

我们可以参考以下的需求变更策略：

（1）所有需求变更必须遵循变更控制过程。

（2）对于未获得批准的变更，不应该做设计和实现工作。

（3）变更应该由项目变更控制委员会决定实现哪些变更。

（4）项目风险承担者应该能够了解变更数据库的内容。

（5）决不能从数据库中删除或者修改变更请求的原始文档。

（6）每一个集成的需求变更必须能跟踪到一个经核准的变更请求。

在实际中，人们总是希望使用自动工具来执行变更控制过程。有许多人使用商业问题跟踪工具来收集、存储、管理需求变更。可以使用工具对一系列最近提交的变更建议产生一个列表给变更控制委员会开会时做议程用。问题跟踪工具也可以随时按变更状态分类报告变更请求的数目。

挑选工具时可以考虑以下几个方面：

（1）可以定义变更请求的数据项。

（2）可以定义变更请求生存期的状态转换图。

（3）可以加强状态转换图使经授权的用户仅能做出所允许的状态变更。

（4）记录每一种状态变更的数据，确认做出变更的人员。

（5）可以定义在提交新请求或请求状态被更新后应该自动通知的设计人员。

（6）可以根据需要生成标准的或定制的报告和图表。

2．变更控制委员会

变更控制委员会可以帮助我们很好地管理项目，哪怕是一个小项目。一个有效率的变更控制委员会定期地考虑每个变更请求，并且基于由此带来的影响和获益做出及时地决策。

变更控制委员会只要能决定合适的人做正确的事就足够了，不必追求大而全。变更控制委员会负责决定哪些已建议需求变更或者新产品特性付诸应用，决定在哪些版本中纠正哪些错误。广义上，变更控制委员会对项目中任何基线工作产品的变更都可以做出决定，需求变更文档仅是其中之一。

变更控制委员会可以由一个小组担任，也可以由多个不同的组担任。变更控制委员会的成员应能代表变更涉及的团体。变更控制委员会可能包括如下方面的代表：

（1）产品或计划管理部门。

（2）项目管理部门。

（3）开发部门。

（4）测试或质量保证部门。

（5）市场部或客户代表。

（6）制作用户文档的部门。

（7）技术支持部门。

（8）帮助桌面或用户支持热线部门。

（9）配置管理部门。

对于小项目只需几个人充当其中的一些角色就可以，并不一定要面面俱到。组建包含软、硬件两方面的项目的变更控制委员会时，也要包括来自硬件工程、系统工程、制造部门或者硬件质量保证和配置管理的代表。建立变更控制委员会在保证权威性的前提下应尽可能精简人员。

变更控制委员会应该有一个总则，用于描述变更控制委员会的目的、授权范围、成员构成、做出决策的过程及操作步骤。总则也应该说明举行会议的频度和事由。管理范围描述该委员会能做什么样的决策，以及有哪一类决策应上报到高一级的委员会。过程及操作步骤如下。

1）制定决策

制定决策过程的描述应确认：

- 变更控制委员会必须到会的人数或做出有效决定必须出席的人数。
- 决策的方法（例如投票，一致通过或其他机制）。

- 变更控制委员会主席是否可以否决该集体的决定。

变更控制委员会应该对每个变更权衡利弊后做出决定。“利”包括节省的资金或额外的收入、增强的客户满意度、竞争优势、减少上市时间；“弊”是指接受变更后产生的负面影响，包括增加的开发费用、推迟的交付日期、产品质量的下降、减少的功能、用户不满意。如果估计的费用超过了本级变更控制委员会的管理范围，应上报到高一级的委员会，否则用制订的决策过程来对变更做出决定。

2）交流情况

一旦变更控制委员会做出决策，指派的人员应及时更新数据库中请求的状态。有的工具可以自动通过电子邮件来通知相关人员。若没有这样的工具，就应该人工通知，以保证他们能充分处理变更。

3）重新协商约定

变更总是有代价的。即使拒绝的变更也因为决策行为（提交、评估、决策）而耗费了资源。变更对新的产品特性会有很大的影响。如果对一个工程项目增加很多新功能，又要求在原先确定的进度计划、人员安排、资金预算和质量要求限制内完成整个项目是不现实的。

当工程项目接受了重要的需求变更时，为了适应变更情况要与管理部门和客户重新协商约定。协商的内容可能包括推迟交货时间、要求增加人手、推迟实现尚未实现的较低优先级的需求，或者质量上进行折中。要是不能获得一些约定的调整，应该把面临的风险写进风险管理计划中。

4.2.5 需求跟踪

需求跟踪包括编制每个需求同系统元素之间的联系文档，这些元素包括别的需求、体系结构、其他设计部件、源代码模块、测试、帮助文件和文档等。跟踪能力信息使变更影响分析十分便利，有利于确认和评估实现某个建议的需求变更所必须的工作。

跟踪能力（联系）链（traceability link）可以使我们跟踪一个需求使用期限的全过程，也就是从初始需求到实现的前后生存期。跟踪能力是优秀需求规格说明书的一个特征，为了实现可跟踪能力，必须统一地标识出每一个需求，以便能明确地进行查阅。

需求跟踪能力链有 4 类，如图 4-9 所示。

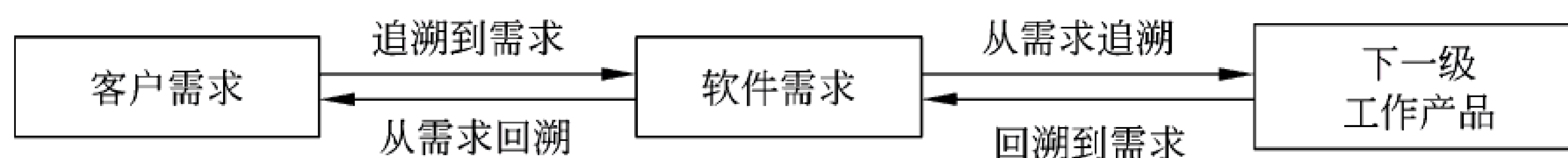


图 4-9 需求可跟踪能力

这 4 类需求跟踪能力链如下：

- 客户需求向前追溯到软件需求。这样就能区分出开发过程中或者开发结束后，由

于客户需求变更受到影响的软件需求。这也就可以确保软件需求规格说明包括所有客户需求。

- 从软件需求回溯相应的客户需求。这也就是确认每个软件需求的源头。如果用使用实例的形式来描述客户需求，那么客户需求与软件需求之间的跟踪情况就是使用实例和功能性需求。
- 从软件需求向前追溯到下一级工作产品。由于开发过程中系统需求转变为软件需求、设计、编码等，所以通过定义单个需求和特定的产品元素之间的（联系）链，可以从需求向前追溯到下一级工作产品。这种联系链告诉我们每个需求对应的产品部件（构件），从而确保产品部件满足每个需求。
- 从产品部件回溯到软件需求。说明了每个部件存在的原因。如果不能把设计元素、代码段或测试回溯到一个需求，就可能存在“画蛇添足”的程序。然而，如果这些孤立的元素表明了一个正当的功能，则说明需求规格说明书漏掉了一项需求。

跟踪能力联系链记录了单个需求之间的父层、互连和依赖的关系。当某个需求变更（被删除或修改）后，这种信息能够确保正确的变更传播，并将相应的任务做出正确的调整。一个项目不必拥有所有种类的跟踪能力联系链，要根据具体的情况调整。

4.2.6　需求变更的代价和风险

“变更是免费的”这种误解是造成项目范围延伸的主要原因之一。人们往往只有在知道变更的成本后才能做出理智的选择。变更需求是要付出代价的，只要允许软件需求变更或者添加新特性，一个表面上很简单的变更也可能转变成很复杂的局面。

大部分的开发人员会遇到添加“没有代价且不影响进度的变更”的要求。没有人愿意做一个费时费力还要担心意想不到的需求变更的事情。需求变更对软件的进度、成本、技术和效率都会有不同程度的影响，变更只能在项目时间、预算、资源等的限制内进行协商。

影响分析是需求管理的一个重要组成部分。影响分析可以提供对建议的变更的准确理解，帮助做出信息量充分的变更批准决策。通过对变更内容的检验，确定对现有的系统做出是修改或者抛弃的决定；或者创建新系统以及评估每个任务的工作量。进行影响分析的能力依赖于跟踪能力、数据的质量和完整性。

4.3　开发管理

4.3.1　项目的范围、时间、成本

1．范围

在初步项目范围说明书中已文档化的主要的可交付物、假设和约束条件的基础上准

备详细的项目范围说明书，是项目成功的关键。在项目规划中，知道了更多的项目信息，项目范围应被更详细地进行描述。应检查假设和约束条件的完整性，并根据需要增加必要的补充假设和约束条件。项目团队和其他对项目范围有不同见解的与项目相关的人，可以基于此履行和准备项目分析。

范围定义的输入包括以下内容。

（1）项目章程。如果项目章程或初始的范围说明书没有在项目执行组织中使用，同样的信息需要进一步收集和开发，以产生详细的项目范围说明书。

（2）项目范围管理计划。

（3）组织过程资产。

（4）批准的变更申请。

经核准的需求变更能引发项目质量、范围、成本或进度的变更。变更申请常常在项目进行过程中被确认，变更申请有多种形式：口头的或书面的，直接的或间接的、外在的或内部的，法律、契约要求的或随意的。

2．时间

项目时间管理包括使项目按时完成所必需的管理过程。进度安排的准确程度可能比成本估计的准确程度更重要。对于成本估计的偏差，软件产品可以靠重新定价或者大量的销售来弥补成本的增加，但如果进度计划不能得到实施则会导致市场机会的丧失或者用户不满意，而且会使成本增加。因此在考虑进度安排时要把人员的工作量与花费的时间联系起来，合理分配工作量，利用进度安排的有效分析方法来严密监视项目的进展情况，以使项目的进度不被拖延。

项目时间管理中的过程包括：活动定义、活动排序、活动的资源估算、活动历时估算、制订进度计划以及进度控制。

为了得到工作分解结构（Work Breakdown Structure，WBS）中最底层的交付物，必须执行一系列的活动。对这些活动的识别以及归档的过程就叫做活动定义。毋庸置疑，定义这些活动的最终目的是为了完成项目的目标。

项目提出后比较明确的一般只是项目的目标，为使项目目标得以实现并且制订出比较完善的项目进度计划，则在对项目进行分解的基础上还必须对分解出的具体目标进行定义，只有这样才能使目标更明确，因此对活动定义是非常必要的。

3．成本

项目成本管理是项目管理的一个重要组成部分，它是指在项目的实施过程中，为了保证完成项目所花费的实际成本不超过其预算成本而展开的项目成本估算、项目预算编制和项目成本控制等方面的管理活动。它包括在批准的预算内完成项目所需要的诸过程，主要有如下一些。

- 成本估算：编制一个为完成项目各活动所需要的资源成本的近似估算。
- 成本预算：将总的成本估算分配到各项活动和工作包上，来建立一个成本的基线。

- 成本控制：控制项目预算的变更。

虽然这里的各个过程是彼此独立、相互间有明确界定的，但在实践中，它们可能会交叉重迭，互相影响，同时与其他知识领域的过程也相互作用。

项目成本管理首先关心的是完成项目活动所需资源的成本，但也应考虑项目决策对项目产品成本的影响。如限制设计审查次数可以降低项目成本，但可能增加顾客的运营成本。项目成本管理的这种广义观点常被称为“全生命周期成本计算”。

在许多应用领域，对项目产品的财务经营状况的预测和分析是在项目之外进行的。但在某些领域（如资金筹措项目），项目成本管理也包括这一工作，这种情况下，项目成本管理将包括一些附加的过程和管理技术，如投资回报、折算现金流、投资回收分析等。

项目成本管理应该考虑项目干系人的信息需求，不同的项目干系人会在不同的时间，以不同的方式检查项目成本。如：采购物品的成本可能在决策结束、订购、发货、收货或会计记账时检查。

为保证项目能够完成预定的目标，必须要加强对项目实际发生成本的控制，一旦项目成本失控，就很难在预算内完成项目，不良的成本控制常常会使项目处于超出预算的危险境地。可是在项目的实际实施过程中，项目超预算的现象还是屡见不鲜。

4.3.2　配置管理、文档管理

1．配置管理

配置管理在项目管理中具有重要的地位和作用。近年来，信息系统项目的规模越来越大，复杂性越来越高；管理上的失误给我们的教训也越来越深刻。这都使得人们不得不重视配置管理问题。

配置管理是 PMBOK、ISO9000 和 CMMI 中的重要组成元素，它在产品开发的生命周期中，提供了结构化的、有序化的、产品化的管理方法，是项目管理的基础工作。配置管理是通过技术和行政手段对产品及其开发过程和生命周期进行控制、规范的一系列措施和过程。信息系统开发过程中的变更以及相应的返工会对产品的质量有很大的影响。如果不从配置管理方面加以控制，必将导致严重的后果。配置管理的一个重要内容就是对变更加以控制，使变更对成本、工期和质量的影响降到最小。

产品配置是指一个产品在其生命周期各个阶段所产生的各种形式（机器可读或人工可读）和各种版本的文档、计算机程序、部件及数据的集合。该集合中的每一个元素称为该产品配置中的一个配置项（Configuration Item，CI），配置项主要有以下两大类。

- 属于产品组成部分的工作成果，如需求文档、设计文档、源代码和测试用例等。
- 属于项目管理和机构支撑过程域产生的文档，如工作计划、项目质量报告、项目跟踪报告等。这些文档虽然不是产品的组成部分，但是值得保存。

每个配置项的主要属性有名称、标识符、文件状态、版本、作者和日期等。所有配置项都被保存在配置库里，确保不会混淆、丢失。配置项及其历史记录反映了项目产品

的演化过程。

置于配置管理之下的工作产品包括将交付给顾客的产品、指定的内部工作产品、采办的产品、工具和其他用于创建和描述这些工作产品的实体。

2．文档管理

文档是影响软件可维护性的决定因素。由于长期使用的大型软件系统在使用过程中必然会经受多次修改，所以文档比程序代码更重要。

软件系统的文档可以分为用户文档和系统文档两类。用户文档主要描述系统功能和使用方法，并不关心这些功能是怎样实现的；系统文档描述系统设计、实现和测试等各方面的内容。

总的说来，软件文档应该满足下述要求：

（1）必须描述如何使用这个系统，没有了这种描述即使是最简单的系统也无法使用。

（2）必须描述怎样安装和管理这个系统。

（3）必须描述系统需求和设计。

（4）必须描述系统的实现和测试，以便使系统成为可维护的。

下面分别讨论用户文档和系统文档。

1）用户文档

用户文档是用户了解系统的第一步，它可以让用户获得对系统的准确的初步印象。

用户文档至少应该包括下述 5 方面的内容。

（1）功能描述：说明系统能做什么。

（2）安装文档：说明怎样安装这个系统以及怎样使系统适应特定的硬件配置。

（3）使用手册：简要说明如何着手使用这个系统（通过丰富的例子说明怎样使用常用的系统功能，并说明用户操作错误时怎样恢复和重新启动）。

（4）参考手册：详尽描述用户可以使用的所有系统设施以及它们的使用方法，并解释系统可能产生的各种出错信息的含义（对参考手册最主要的要求是完整，因此通常使用形式化的描述技术）。

（5）操作员指南（如果需要有系统操作员的话）：说明操作员应如何处理使用中出现的各种情况。

2）系统文档

所谓系统文档指从问题定义、需求说明到验收测试计划这样一系列和系统实现有关的文档。描述系统设计、实现和测试的文档对于理解程序和维护程序来说是非常重要的。

4.3.3 软件开发的质量与风险

1．软件质量

成功的项目管理是在约定的时间和范围、预算的成本以及要求的质量下，达到项目干系人的期望。能否成功地管理一个项目，质量的好坏也非常重要。质量管理是项目管

理的重要方面之一，它与范围、成本和时间是项目成功的关键因素。项目质量管理包括为确保项目能够满足所要执行的需求的过程，包括质量管理职能的所有活动。

ISO9000 对项目质量的定义是：一组固有特性满足需求的程度。需求指明示的、通常隐含的或必须履行的需求或期望。特性是指可区分的特征，可以是固有的或赋予的、定性的或定量的、有各种类别（物理的、感官的、行为的、时间的、功能的等）。

美国质量管理协会把质量定义为“过程、产品或服务满足明确或隐含的需求能力的特征”。质量与范围、成本和时间是项目成功的关键因素。明确和隐含的需求是制定项目需求的输入。在项目领域，质量管理的一个关键因素是通过项目范围管理转换隐含需求为项目需求。不要把质量和等级相混淆，等级是指具有相同使用功能不同技术特性的产品或服务的类别。质量低说明产品或服务存在问题，没有达到要求，而等级低的产品或服务就不一定存在问题。

2．软件开发风险

项目是在复杂的自然和社会环境中进行的，受众多因素的影响。对于这些内外因素，从事项目活动的主体往往认识不足或者没有足够的力量加以控制。项目的过程和结果常常出乎人们的意料，有时不但未达到项目主体预期的目的，反而使其蒙受各种各样的损失；而有时又会给他们带来很好的机会。项目同其他经济活动一样带有风险。要避免和减少损失，将威胁化为机会，项目主体就必须了解和掌握项目风险的来源、性质和发生规律，进而施行有效的管理。

对项目风险进行管理，国际上已经成为项目管理的重要方面。如世界银行对每一个贷款项目都进行风险分析，制定风险管理计划，写在有关的文件之中，并付诸行动。

在项目所处的自然、经济、社会和政治环境中，每一个项目都有风险。完全避开或消除风险，或者只享受权益而不承担风险，是不可能的。另一方面，对项目风险进行认真的分析、科学的管理，是能够避开不利条件、少受损失、取得预期的结果并实现项目目标的。

当事件、活动或项目有损失或收益与之相联系，涉及到某种或然性或不确定性和涉及到某种选择时，才称为有风险。以上三条，每一个都是风险定义的必要条件，不是充分条件。具有不确定性的事件不一定是风险。

项目风险是一种不确定的事件或条件，一旦发生，会对项目目标产生某种正面或负面的影响。风险有其成因，同时，如果风险发生，也导致某种后果。举例来说，风险成因可能是需要获取某种许可，或是项目的人力资源受到限制。风险事件本身则是获取许可所花费的时间可能比计划的要长，或是可能没有充足的人员来完成项目工作。以上任何一种不确定事件一旦发生，都会给项目的成本、进度计划或质量带来某种后果。风险条件可能包括组织环境中导致项目风险的某些因素，如不良的项目管理，或对不能控制的外部参与方的依赖。

项目风险既包括对项目目标的威胁，也包括促进项目目标的机会。风险源于所有项

目之中的不确定因素。已知风险是那些已经经过识别和分析的风险。对于已知风险，进行相应计划是可能的。虽然项目经理们可以依据以往类似项目的经验，采取一般的应急措施处理未知风险，但未知风险是无法管理的。

4.4 设计方法

4.4.1 结构化分析与设计

结构化程序设计的概念最早由 E. W. Dijkstra 提出，其理由是 GOTO 语句对程序的可读性、可测试性和可维护性带来极大的危害，应该用更可维护的控制结构替代它。随后，Bohm 和 Jacopini 证明了仅用“顺序”、“分支”和“循环”三种基本的控制构件即能构造任何单入口单出口程序，这个结论奠定了结构程序设计的理论基础。

何谓结构程序设计，目前尚无定论，较流行的定义为：结构程序设计是程序设计技术，它采用自顶向下逐步求精的设计方法和单入口单出口的控制构件。

自顶向下逐步求精的方法是人类解决复杂问题时常用的一种方法，采用这种先整体后局部，先抽象后具体的步骤开发的软件一般具有较清晰的层次。此外，由于仅使用单入口单出口的控制构件，使程序有良好的结构特征，这些都能大大降低程序的复杂性，增强程序的可读性、可维护性和可验证性，从而提高软件的生产率。

另一方面，采用结构程序设计的技术可能会多占用一些时间和空间资源，这也是那些反对从高级语言中排除 GOTO 语句者的主要依据。实际上，随着硬件技术的飞速发展，因结构程序设计所增加的这点耗费对大多数应用来说已不再是重要的因素。

结构程序设计的思想应该在软件设计中体现出来，但这并不排除为效率或其他原因对结构程序设计做一点修正。随着面向对象、软件重用等新的软件开发方法和技术的发展，更现实、更有效的开发途径可能是自顶向下和自底向上两种方法有机的结合。

4.4.2 面向对象的分析设计

面向对象的分析模型主要由顶层架构图、用例与用例图、领域概念模型构成；设计模型则包含以包图表示的软件体系结构图、以交互图表示的用例实现图、完整精确的类图、针对复杂对象的状态图和用以描述流程化处理过程的活动图等。为完成这一转换过程，设计人员必须处理以下任务：

（1）针对分析模型中的用例，设计实现方案。实现方案用 UML 交互图表示。

（2）设计技术支撑设施。在大型软件项目中，往往需要一些技术支撑设施来帮助业务需求层面的类或子系统完成其功能。这些设施本身并非业务需求的一部分，但却为多种业务需求的实现提供公共服务。例如，数据的持久存储服务、安全控制服务和远程访问服务等。在面向对象设计中，需要研究这些技术支撑设施的实现方式以及它们与业务

需求层面的类及子系统之间的关系。

设计用户界面。针对分析模型中的领域概念模型以及引进的新类，完整、精确地确定每个类的属性和操作，并完整地标示类之间的关系。此外，为了实现软件重用和强内聚、松耦合等软件设计原则，还可以对已经形成的类图进行各种微调，最终形成足以构成面向对象程序设计的基础和依据的详尽类图。面向对象的软件设计过程如图 4-10 所示。

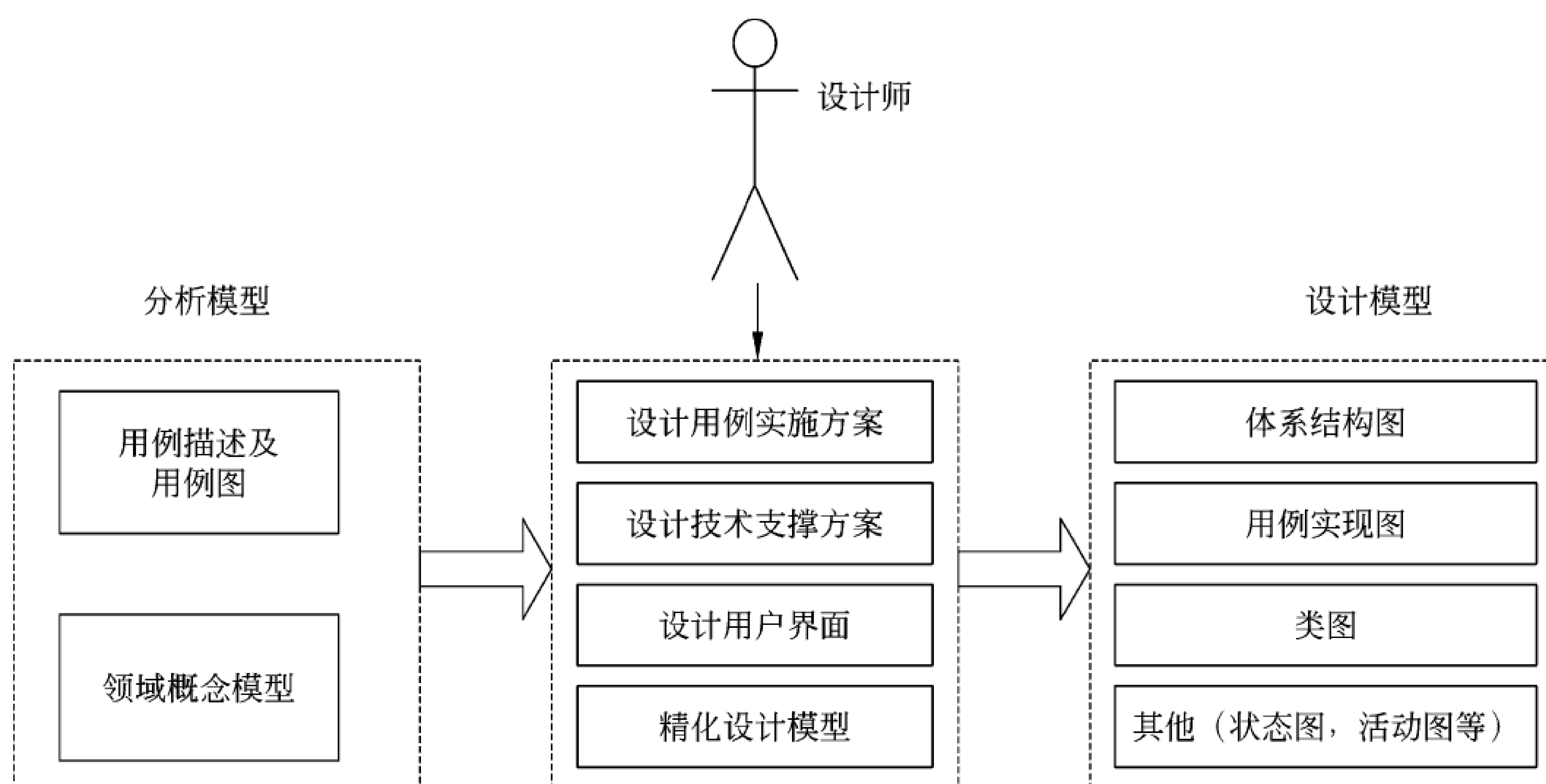


图 4-10　面向对象的软件设计过程

4.5　软件的重用

软件重用是指在两次或多次不同的软件开发过程中重复使用相同或相似软件元素的过程。软件元素包括需求分析文档、设计过程、设计文档、程序代码、测试用例和领域知识等。对于新的软件开发项目而言，它们或者是构成整个目标软件系统的部件，或者在软件开发过程中发挥某种作用。通常将这些软件元素称为软部件。

为了能够在软件开发过程中重用现有的软部件，必须在此之前不断地进行软部件的积累，并将它们组织成软部件库。这就是说，软件重用不仅要讨论如何检索所需的软部件以及如何对它们进行必要的修剪，还要解决如何选取软部件、如何组织软部件库等问题。因此，软件重用方法学通常要求软件开发项目既要考虑重用已有软部件的机制，又要系统地考虑生产可重用软部件的机制。这类项目通常称为软件重用项目。

按照重用活动是否跨越相似性较少的多个应用领域，软件重用可区别为横向重用和纵向重用。横向重用（horizontal reuse）是指重用不同应用领域中的软件元素，例如数据

结构、分类算法和人机界面构件等。标准函数库是一种典型的、原始的横向重用机制。纵向重用是指在一类具有较多公共性的应用领域之间进行软部件重用。因为在两个截然不同的应用领域之间实施软件重用的潜力不大，所以纵向重用才广受瞩目，并成为软件重用技术的真正希望所在。不难理解，纵向重用活动的主要关键点即是域分析：根据应用领域的特征及相似性预测软部件的可重用性。一旦根据域分析确认了软部件的重用价值，即可进行软部件的开发，并对具有重用价值的软部件进行一般化，以便它们能够适应新的类似的应用领域。然后，软部件及其文档即可进入软部件库，成为可供后续开发项目使用的可重用资源。这些步骤构成软部件的构造活动。显然，它是一个软部件不断积累、不断完善的渐进过程。随着软部件的不断丰富，软部件库的规模会不断扩大，因此，库的组织结构将直接影响软部件的检索效率，特别是当检索手段并不局限于标准函数库所采用的简单名字匹配方法时。可供候选的软部件从库中被检索出来以后，用户还必须理解其功能或行为，以判别它是否真正适应于当前项目。必要时可考虑对某个与期望的功能 / 行为匹配程度最佳的软部件进行稍许修改，甚至可以将修改后的软部件加进软部件库以替代原有软部件。当然，这要求修改后的软部件比原有软部件具有更高的重用价值。上述软件重用方法如图 4-11 所示。

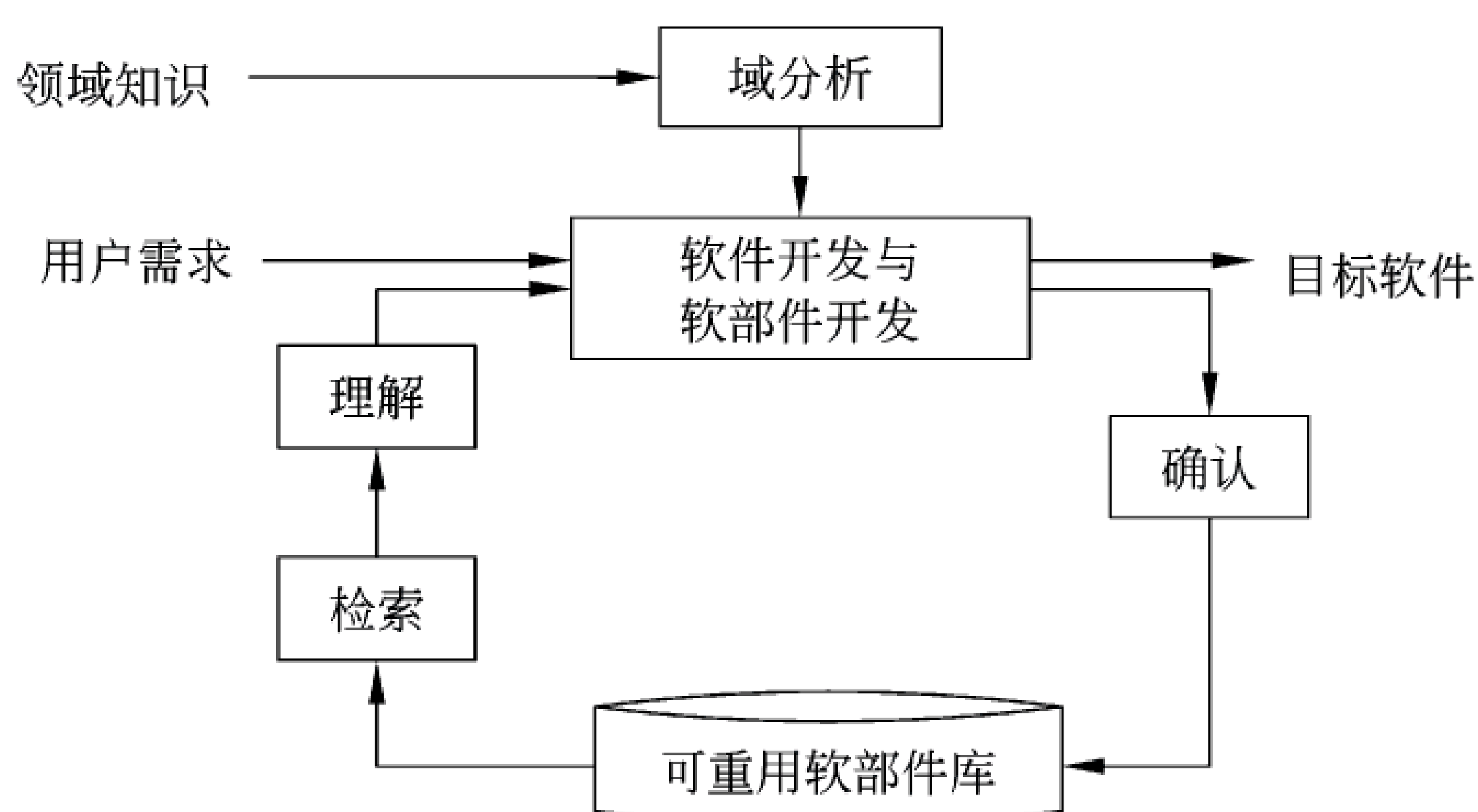


图 4-11　软件重用方法原理

使用重用技术可以减少软件开发活动中大量的重复性工作，这样就能够提高软件生产率，降低开发成本，缩短开发周期。同时，由于软部件大都经过严格的质量认证，并在实际运行环境中得到检验，因此重用软部件有助于改善软件质量。此外，大量使用软部件，软件的软件逆工程灵活性和标准化程度也可望得到提高。

4.6　逆向工程与重构工程

逆向工程与重构工程是目前预防性维护采用的主要技术。逆向工程术语源于硬件制

造业，相互竞争的公司为了了解对方设计和制造工艺的机密，在得不到设计和制造说明书的情况下，通过拆卸实物获取信息，软件的逆向工程也基本类似，不过通常“解剖”的不仅是竞争对手的程序，而且还包括本公司多年前的产品，此时得不到设计“机密”的主要障碍是缺乏文档。因此，所谓软件的逆向工程就是分析已有的程序，寻求比源代码更高级的抽象表现形式。一般认为，凡是在软件生命周期内将软件某种形式的描述转换成更为抽象形式的活动都可称为逆向工程。与之相关的概念是：重构（restructuring），指在同一抽象级别上转换系统描述形式；设计恢复（design recovery），指借助工具从已有程序中抽象出有关数据设计、总体结构设计和过程设计的信息（不一定是原设计）；重构工程（re-engineering），也称修复和改造工程，它是在逆向工程所获信息的基础上修改或重构已有的系统，产生系统的一个新版本。

1．恢复信息的级别

逆向工程导出的信息可分为如下 4 个抽象层次。

（1）实现级：包括程序的抽象语法树、符号表等信息。

（2）结构级：包括反映程序分量之间相互依赖关系的信息，例如调用图、结构图等。

（3）功能级：包括反映程序段功能及程序段之间关系的信息。

（4）领域级：包括反映程序分量或程序诸实体与应用领域概念之间对应关系的信息。

显然，上述信息的抽象级别越高，它与代码的距离就越远，通过逆向工程恢复的难度亦越大，而自动工具支持的可能性相对变小，要求人参与判断和推理的工作增多。

2．恢复信息的方法

在逆向工程中用于恢复信息的方法有 4 类。第一类为用户指导下的搜索与变换（User-Directed Search and Transformation）。此类方法用于导出实现级和结构级信息。它要求维护人员在数据库系统的支持下，运用询问语言，针对源代码或与之相近的表示形式，指定待查找的句型（pattern），根据搜索结果析出所需信息或进行特殊变换。

第二类方法为变换式方法（Transformational Approaches），除领域级外所有抽象级别上的信息都可用此类方法推导。变换式方法又细分为不需要维护人员过多干涉的自动分析法（如静态分析和调用图、控制流图生成等）和基于特定库的用户指导变换法两类。变换方法自动化程度越高，得到的设计信息越粗略，因为任何深层次的分析不可避免地要借助人的智力。一般借助变换法得到程序的某种中间表示形式，通过进一步使用其他工具将已获粗略的设计信息精化为完整、一致的软件设计。

第三类方法是基于领域知识的（Domain Knowledge-Based），主要用于恢复功能级和领域级信息。领域知识一般用规则库表示，用已确定或假定的领域概念与代码之间的对应关系推导进一步的假设，最后导出程序的功能。显然该类方法的不确定性最大，因此目前成熟的工具和原型系统还很少见。

最后一类方法称为铅板恢复法，这类方法仅适用于推导实现级和结构级信息。这些

方法用于识别程序设计“铅板”或公共结构，“铅板”既可为一个简单算法（如两变量互换值），亦可为相对复杂的成分（如冒泡分类）。因铅板与程序之间可能存在多种匹配形式，所以此类方法还包含大量的推理与决策。各类方法采用的输入形式、搜索策略和推理策略都不尽相同。后两类方法又称为基于知识的方法。

尽管每个软件组织都可能有数百万行代码可供重构，但由于缺乏时机和支持工具或者因为经济上得不偿失，往往只有那些决定或移植、或重新设计、或为重用而需验证正确性的程序才被选择实施逆向工程。

第 5 章　软件架构设计

Shaw 和 Garlan 在他们划时代的著作中以如下方式讨论了软件的体系结构：从第一个程序被划分成模块开始，软件系统就有了体系结构。现在，有效的软件体系结构及其明确的描述和设计，已经成为软件工程领域中重要的主题。

由于历史原因，研究者和工程人员对 Software Architecture 简称 SA 的翻译不一样，本书中软件“体系结构”和“架构”具有相同的含义。

5.1　软件架构概念

5.1.1　软件架构的定义

Bass、Clements 和 Kazman 对于这个难懂的概念给出了如下的定义：

一个程序和计算系统软件体系结构是指系统的一个或者多个结构。结构中包括软件的构件，构件的外部可见属性以及它们之间的相互关系。

体系结构并非可运行软件。确切地说，它是一种表达，使软件工程师能够：

（1）分析设计在满足规定需求方面的有效性。

（2）在设计变更相对容易的阶段，考虑体系结构可能的选择方案。

（3）降低与软件构造相关联的风险。

上面的定义强调在任意体系结构表述中“软件构件”的角色。在体系结构设计的环境中，软件构件可以简单到程序模块或者面向对象的类，也可以扩充到包含数据库和能够完成客户与服务器网络配置的“中间件”。

软件体系结构的设计通常考虑了设计金字塔中的两个层次——数据设计和体系结构设计。数据设计使我们表示出传统系统中体系结构的数据构件和面向对象系统中类的定义（封装了属性和操作），体系结构设计则主要关注软件构件的结构、属性和交互作用。

建立体系结构层的“内聚的、良好设计的表示”所需的方法，其目标是提供一种导出体系结构设计的系统化方法，而体系结构设计是构建软件的初始蓝图。

5.1.2　软件架构设计与生命周期

1．需求分析阶段

需求阶段的 SA 研究还处于起步阶段。在本质上，需求和 SA 设计面临的是不同的对象：一个是问题空间；另一个是解空间。保持二者的可追踪性和转换，一直是软件工

程领域追求的目标。从软件需求模型向 SA 模型的转换主要关注两个问题：

（1）如何根据需求模型构建 SA 模型。

（2）如何保证模型转换的可追踪性。

针对这两个问题的解决方案，随着所采用的需求模型的不同而各异。在采用 Use Case 图描述需求的方法中，从 Use Case 图向 SA 模型（包括类图等）的转换一般经过词性分析和一些经验规则来完成，而可追踪性则可通过表格或者 Use Case Map 等来维护。

从软件复用的角度看，SA 影响需求工程也有其自然性和必然性，已有系统的 SA 模型对新系统的需求工程能够起到很好的借鉴作用。在需求阶段研究 SA，有助于将 SA 的概念贯穿整个软件生命周期，从而保证了软件开发过程的概念完整性，有利于各阶段参与者的交流，也易于维护各阶段的可追踪性。

2．设计阶段

设计阶段是 SA 研究关注的最早和最多的阶段，这一阶段的 SA 研究主要包括：SA 模型的描述、SA 模型的设计与分析方法，以及对 SA 设计经验的总结与复用等。有关 SA 模型描述的研究分为三个层次：

（1）SA 的基本概念，即 SA 模型由哪些元素组成，这些组成元素之间按照何种原则组织。传统的设计概念只包括构件（软件系统中相对独立的有机组成部分，最初称为模块）以及一些基本的模块互联机制。随着研究的深入，构件间的互联机制逐渐独立出来，成为与构件同等级别的实体，称为连接子。现阶段的 SA 描述方法是构件和连接子的建模。近年来，也有学者认为应当把 Aspect 等引入 SA 模型。

（2）体系结构描述语言（Architecture Description Language，ADL），支持构件、连接子及其配置的描述语言就是如今所说的体系结构描述语言。ADL 对连接子的重视成为区分 ADL 和其他建模语言的重要特征之一。典型的 ADL 包括 UniCon、Rapide、Darwin、Wright、C2 SADL、Acme、xADL、XYZ/ADL 和 ABC/ADL 等。

（3）SA 模型的多视图表示，从不同的视角描述特定系统的体系结构，从而得到多个视图，并将这些视图组织起来以描述整体的 SA 模型。多视图作为一种描述 SA 的重要途径，也是近年来 SA 研究领域的重要方向之一。系统的每一个不同侧面的视图反映了一组系统相关人员所关注的系统的特定方面，多视图体现了关注点分离的思想。

把体系结构描述语言和多视图结合起来描述系统的体系结构，能使系统更易于理解，方便系统相关人员之间进行交流，并且有利于系统的一致性检测以及系统质量属性的评估。学术界已经提出若干多视图的方案，典型的包括 4+1 模型（逻辑视图、进程视图、开发视图、物理视图，加上统一的场景）、Hofmesiter 的 4 视图模型（概念视图、模块视图、执行视图、代码视图）、CMU-SEI 的 Views and Beyond 模型（模块视图、构件和连接子视图、分配视图）等。此外，工业界也提出了若干多视图描述 SA 模型的标准，如 IEEE 标准 1471-2000（软件密集型系统体系结构描述推荐实践）、开放分布式处理参考模型（RM-ODP）、统一建模语言（UML）以及 IBM 公司推出的 Zachman 框架等。需

要说明的是，现阶段的 ADL 大多没有显式地支持多视图，并且上述多视图并不一定只描述设计阶段的模型。

3．实现阶段

最初的 SA 研究往往只关注较高层次的系统设计、描述和验证。为了有效实现从 SA 设计向实现的转换，实现阶段的体系结构研究在以下几个方面。

（1）研究基于 SA 的开发过程支持，如项目组织结构、配置管理等。

（2）寻求从 SA 向实现过渡的途径，如将程序设计语言元素引入 SA 阶段、模型映射、构件组装、复用中间件平台等。

（3）研究基于 SA 的测试技术。

SA 提供了待生成系统的蓝图，根据该蓝图实现系统需要较好的开发组织结构和过程管理技术。以体系结构为中心的软件项目管理方法，开发团队的组织结构应该和体系结构模型有一定的对应关系，从而提高软件开发的效率和质量。

对于大型软件系统而言，由于参与实现的人员较多，所以需要提供适当的配置管理手段。SA 引入能够有效扩充现有配置管理的能力，通过在 SA 描述中引入版本、可选择项（options）等信息，可以分析和记录不同版本构件和连接子之间的演化，从而可用来组织配置管理的相关活动。典型的例子包括支持给构件指定多种实现的 UniCon、支持给构件和连接子定义版本信息和可选信息的 xADL 等。

为了填补高层 SA 模型和底层实现之间的鸿沟，通过封装底层的实现细节，模型转换、精化等手段缩小概念之间的差距。典型的方法如下。

（1）在 SA 模型中引入实现阶段的概念，如引入程序设计语言元素等。

（2）通过模型转换技术，将高层的 SA 模型逐步精化成能够支持实现的模型。

（3）封装底层的实现细节，使之成为较大粒度构件，在 SA 指导下通过构件组装的方式实现系统，这往往需要底层中间件平台的支持。

4．构件组装阶段

在 SA 设计模型的指导下，可复用构件组装可以在较高层次上实现系统，并能够提高系统实现的效率。在构件组装的过程中，SA 设计模型起到了系统蓝图的作用。研究内容包括：

（1）如何支持可复用构件的互联，即对 SA 设计模型中规约的连接子的实现提供支持。

（2）在组装过程中，如何检测并消除体系结构失配问题。

对设计阶段连接子的支持：不少 ADL 支持在实现阶段将连接子转换到具体的程序代码或系统实现，如 UniCon 定义了 Pipe、FileIO、ProcedureCall 等多种内建的连接子类型，它们在设计阶段被实例化，并可以在实现阶段在工具的支持下转化成为具体的实现机制，如过程调用、操作系统数据访问、Unix 管道和文件、远程过程调用等。支持从 SA 模型生成代码的体系结构描述语言，如 C2 SADL、Rapide 等，也都提供了一定的机

制生成连接子的代码。

中间件遵循特定的构件标准，为构件互联提供支持，并提供相应的公共服务，如安全服务、命名服务等。中间件支持的连接子实现有如下优势：

（1）中间件提供了构件之间跨平台交互的能力，且遵循特定的工业标准，如 CORBA、J2EE、COM 等，可以有效地保证构件之间的通信完整性。

（2）产品化的中间件可以提供强大的公共服务能力，这样能够更好地保证最终系统的质量属性。设计阶段连接子的规约可以用于中间件的选择，如消息通信连接子最好选择提供消息通信机制的中间件平台。从某种意义上说，随着中间件技术的发展，也导致一类新的 SA 风格，即中间件导向的体系结构风格（middleware-induced architectural style）的出现。

检测并消除体系结构失配：体系结构失配问题是由 David Garlan 等人在 1995 年提出。失配是指在软件复用的过程中，由于待复用构件对最终系统的体系结构和环境的假设（assumption）与实际状况不同而导致的冲突。在构件组装阶段失配问题主要包括：

（1）由构件引起的失配，包括由于系统对构件基础设施、构件控制模型和构件数据模型的假设存在冲突引起的失配。

（2）由连接子引起的失配，包括由于系统对构件交互协议、连接子数据模型的假设存在冲突引起的失配。

（3）由于系统成分对全局体系结构的假设存在冲突引起的失配等。要解决失配问题，首先需要检测出失配问题，并在此基础上通过适当的手段消除检测出的失配问题。

5．部署阶段

随着网络与分布式软件的发展，软件部署逐渐从软件开发过程中独立出来，成为软件生命周期中一个独立的阶段。为了使分布式软件满足一定的质量属性要求，如性能、可靠性等，部署需要考虑多方面的信息，如待部署软件构件的互联性、硬件的拓扑结构、硬件资源占用（如 CPU、内存）等。SA 对软件部署作用如下。

（1）提供高层的体系结构视图描述部署阶段的软硬件模型。

（2）基于 SA 模型可以分析部署方案的质量属性，从而选择合理的部署方案。

现阶段，基于 SA 的软件部署研究更多地集中在组织和展示部署阶段的 SA、评估分析部署方案等方面，部署方案的分析往往停留在定性的层面，并需要部署人员的参与。

6．后开发阶段

后开发阶段是指软件部署安装之后的阶段。这一阶段的 SA 研究主要围绕维护、演化、复用等方面来进行。典型的研究方向包括动态软件体系结构、体系结构恢复与重建等。

1）动态软件体系结构

传统的 SA 研究设想体系结构总是静态的，即软件的体系结构一旦建立，就不会在运行时刻发生变动。但人们在实践中发现，现实中的软件往往具有动态性，即它们的体

系结构会在运行时发生改变。SA 在运行时发生的变化包括两类。一类是软件内部执行所导致的体系结构改变。比如，很多服务器端软件会在客户请求到达时创建新的构件来响应用户的需求。某个自适应的软件系统可能根据不同的配置状况采用不同的连接子来传送数据。另一类变化是软件系统外部的请求对软件进行的重配置。比如，有很多高安全性的软件系统，这些系统在升级或进行其他修改时不能停机。因为修改是在运行时刻进行的，体系结构也就动态地发生了变化。在高安全性系统之外也有很多软件需要进行动态修改，比如很多操作系统期望能够在升级时无须重新启动系统，在运行过程中就完成对体系结构的修改。

由于软件系统会在运行时刻发生动态变化，这就给体系结构的研究提出了很多新的问题。如何在设计阶段捕获体系结构的这种动态性，并进一步指导软件系统在运行时刻实施这些变化，从而达到系统的在线演化或自适应甚至自主计算，是动态体系结构所要研究的内容。现阶段，动态软件体系结构研究可分为以下两个部分。

（1）体系结构设计阶段的支持。主要包括变化的描述、根据变化如何生成修改策略、描述修改过程、在高抽象层次保证修改的可行性以及分析、推理修改所带来的影响等。

（2）运行时刻基础设施的支持。主要包括系统体系结构的维护、保证体系结构修改在约束范围内、提供系统的运行时刻信息、分析修改后的体系结构符合指定的属性、正确映射体系结构构造元素的变化到实现模块、保证系统的重要子系统的连续执行并保持状态、分析和测试运行系统等。

2）体系结构恢复与重建

当前系统的开发很少是从头开始的，大量的软件开发任务是基于已有的遗产系统进行升级、增强或移植。这些系统在开发的时候没有考虑 SA，在将这些系统进行构件化包装、复用的时候，会得不到体系结构的支持。因此，从这些系统中恢复或重构体系结构是有意义的，也是必要的。

SA 重建是指从已实现的系统中获取体系结构的过程。一般地，SA 重建的输出是一组体系结构视图。现有的体系结构重建方法可以分为 4 类：

（1）手工体系结构重建。

（2）工具支持的手工重建。通过工具对手工重建提供辅助支持，包括获得基本体系结构单元、提供图形界面允许用户操作 SA 模型、支持分析 SA 模型等。如 KLOCwork inSight 工具（www.klocwork.com/products/insight.asp）使用代码分析算法直接从源代码获得 SA 构件视图，用户可以通过操作图形化的 SA 设定体系结构规则，并可在工具的支持下实现对体系结构的理解、自动控制和管理。

（3）通过查询语言来自动建立聚集。这类方法适用于较大规模的系统，基本思路是：在逆向工程工具的支持下分析程序源代码，然后将所得到的体系结构信息存入数据库，并通过适当的查询语言得到有效的体系结构显示。

（4）使用其他技术，比如数据挖掘等。

5.1.3 软件架构的重要性

软件架构设计是降低成本、改进质量、按时和按需交付产品的关键因素。

1．架构设计能够满足系统的品质

系统的功能性是软件构架师通过组成体系架构的多种元素之间的交互作用来支持的。架构设计用于实现系统的品质，如性能、安全性和可维护性等。通过架构设计文档化，可以尽早的评估项目的这些品质。

2．架构设计使受益人达成一致的目标

架构设计的过程使得不同的受益人达成一致的目标，体系架构的过程需要确保架构设计被清楚地传达与理解。一个被有效传达的体系架构使得涉众们可以辩论决议和权衡，反复讨论，最终达成共识。文档化体系架构是非常重要的，这是软件构架师的主要职责。

3．架构设计能够支持计划编制过程

架构设计将确定组件之间的依赖关系，直接支持项目计划和项目管理的活动，例如，细节化分，日程安排，工作分配，成本分析，风险管理和技能开发等；构架师还能协助估算项目成本，例如，体系架构决定使用第三方组件的成本，以及支持开发的所有工具的成本；构架师支持技术风险的管理，包括制定每一个风险的优先次序，以及确定一个恰当的风险缓解策略。

4．架构设计对系统开发的指导性

架构设计主要目标就是确保体系架构能够为设计人员和实现人员所承担的工作提供可靠的框架。很明显，这比简单的传送一个体系架构视图要复杂的多。为了确保最终体系架构的完整性，构架师必须明确的定义体系架构，因为它确定了体系架构的重要元素，例如系统的组件，组件之间的接口以及组件之间的通信。

构架师同时还必须定义恰当的标准和指导方针，它们将会引导设计人员和实现人员的工作。对开发过程活动采取恰当的架构回顾和评估，能够确保体系架构的完整性。这些 QA（Quality Assurance，质量保障）活动的任务是确定体系架构的标准和指导方针的有效性。

5．架构设计能够有效地管理复杂性

如今的系统越来越复杂，这种复杂性需要我们去管理。体系架构通过构件及构件之间关系，描述了一个抽象的系统，因而提供了高层次的复杂管理的方法。同样，架构设计过程考虑组件的递归分解。这是处理一个大的问题的很好的一个方法，它可以把这个大问题分解成很多的小问题，再逐个的解决。

6．架构设计为复用奠定了基础

架构设计过程可以同时支持使用和建立复用资源。复用资源可以降低一个系统的成本，并且可以改进系统的质量，这些好处已经被证明。一个体系架构的建立，能够支持大粒度的资源复用。例如，体系架构的重要组件和它们之间的接口和质量，能够支持现货供应的组件，存在的系统和封装的应用程序等的选择，从而可以用来实现这些组件。

7．架构设计能够降低维护费用

架构设计过程可以在很多方面帮助我们降低维护费用。首先最重要的是架构设计过程要确保系统的维护人员是一个主要的涉众，并且他们的需求被作为首要的任务满足。一个被恰当文档化的体系架构不应该仅仅为了减轻系统的可维护性；构架师还应该确保结合了恰当的系统维护机制，并且在建立体系架构的时候还要考虑系统的适应性和可扩充性。

8．架构设计能够支持冲突分析

架构设计的一个重要的好处是它可以允许我们在采取改变之前推断它所产生的影响。一个软件构架确定了主要的组件和它们之间的交互作用，两个组件之间的依赖性以及这些组件对于需求的可追溯性。有了这个信息，例如需求的改变等可以通过组件的影响来分析。同样的，改变一个组件的影响可以在依靠它的其他组件上分析出来。

5.2　基于架构的软件开发方法

5.2.1　体系结构的设计方法概述

基于体系结构的软件设计（Architecture-Based Software Design，ABSD）方法。ABSD 方法是体系结构驱动，即指构成体系结构的商业、质量和功能需求的组合驱动的。使用 ABSD 方法，设计活动可以从项目总体功能框架明确就开始，这意味着需求抽取和分析还没有完成（甚至远远没有完成），就开始了软件设计。设计活动的开始并不意味着需求抽取和分析活动就可以终止，而是应该与设计活动并行。特别是在不可能预先决定所有需求时，例如产品线系统或长期运行的系统，快速开始设计是至关重要的。

ABSD 方法有三个基础。第一个基础是功能的分解。在功能分解中，ABSD 方法使用已有的基于模块的内聚和耦合技术。第二个基础是通过选择体系结构风格来实现质量和商业需求。第三个基础是软件模板的使用。软件模板利用了一些软件系统的结构。

ABSD 方法是递归的，且迭代的每一个步骤都是清晰地定义的。因此，不管设计是否完成，体系结构总是清晰的，这有助于降低体系结构设计的随意性。

5.2.2　概念与术语

1．设计元素

ABSD 方法是一个自顶向下，递归细化的方法，软件系统的体系结构通过该方法得到细化，直到能产生软件构件和类。

ABSD 方法中使用的设计元素如图 5-1 所示。在最顶层，系统被分解为若干概念子系统和一个或若干个软件模板。在第二层，概念子系统又被分解成概念构件和一个或若干个附加软件模板。

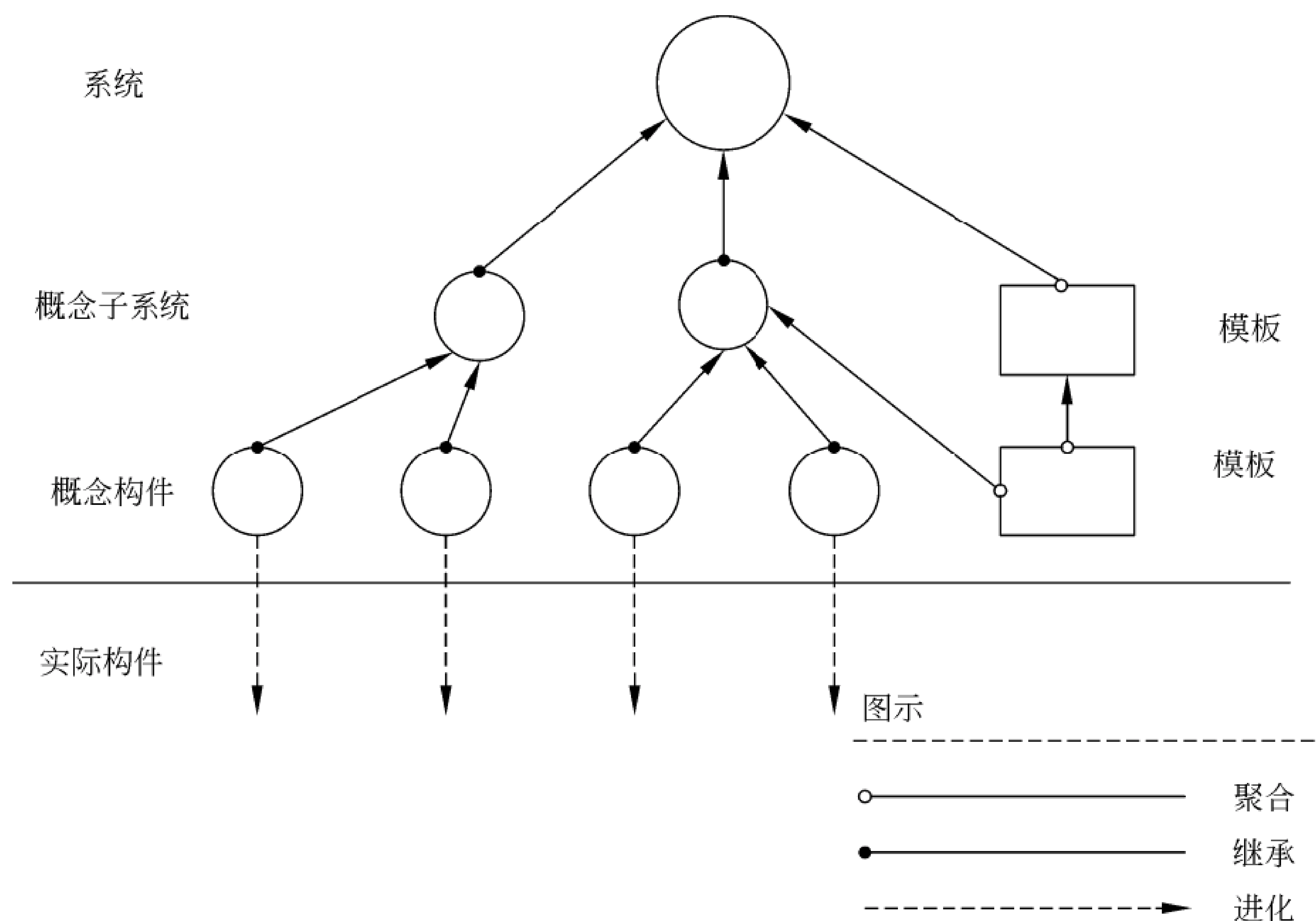


图 5-1 ABSD 方法过程

2．视角与视图

考虑体系结构时，重要的是从不同的视角（perspective）来检查，这促使软件设计师考虑体系结构的不同属性。例如，展示功能组织的静态视角能判断质量特性，展示并发行为的动态视角能判断系统行为特性。选择的特定视角或视图也就是逻辑视图、进程视图、实现视图和配置视图。使用逻辑视图来记录设计元素的功能和概念接口，设计元素的功能定义了它本身在系统中的角色，这些角色包括功能性能等。

3．用例和质量场景

用例已经成为推测系统在一个具体设置中的行为的重要技术，用例被用在很多不同的场合，用例是系统的一个给予用户一个结果值的功能点，用例用来捕获功能需求。

在使用用例捕获功能需求的同时，我们通过定义特定场景来捕获质量需求，并称这些场景为质量场景。这样一来，在一般的软件开发过程中，我们使用质量场景捕获变更、性能、可靠性和交互性，分别称之为变更场景、性能场景、可靠性场景和交互性场景。质量场景必须包括预期的和非预期的。例如，一个预期的性能场景是估计每年用户数量增加 10％的影响，一个非预期的场景是估计每年用户数量增加 100％的影响。非预期场景可能不能真正实现，但它们在决定设计的边界条件时很有用。

5.2.3 基于体系结构的开发模型

本节讨论基于体系结构的软件开发模型。 传统的软件开发过程可以划分为从概念

直到实现的若干个阶段，包括问题定义、需求分析、软件设计、软件实现及软件测试等。如果采用传统的软件开发模型，软件体系结构的建立应位于需求分析之后，概要设计之前。

传统软件开发模型存在开发效率不高，不能很好地支持软件重用等缺点。ABSDM 模型把整个基于体系结构的软件过程划分为体系结构需求、设计、文档化、复审、实现和演化等 6 个子过程，如图 5-2 所示。

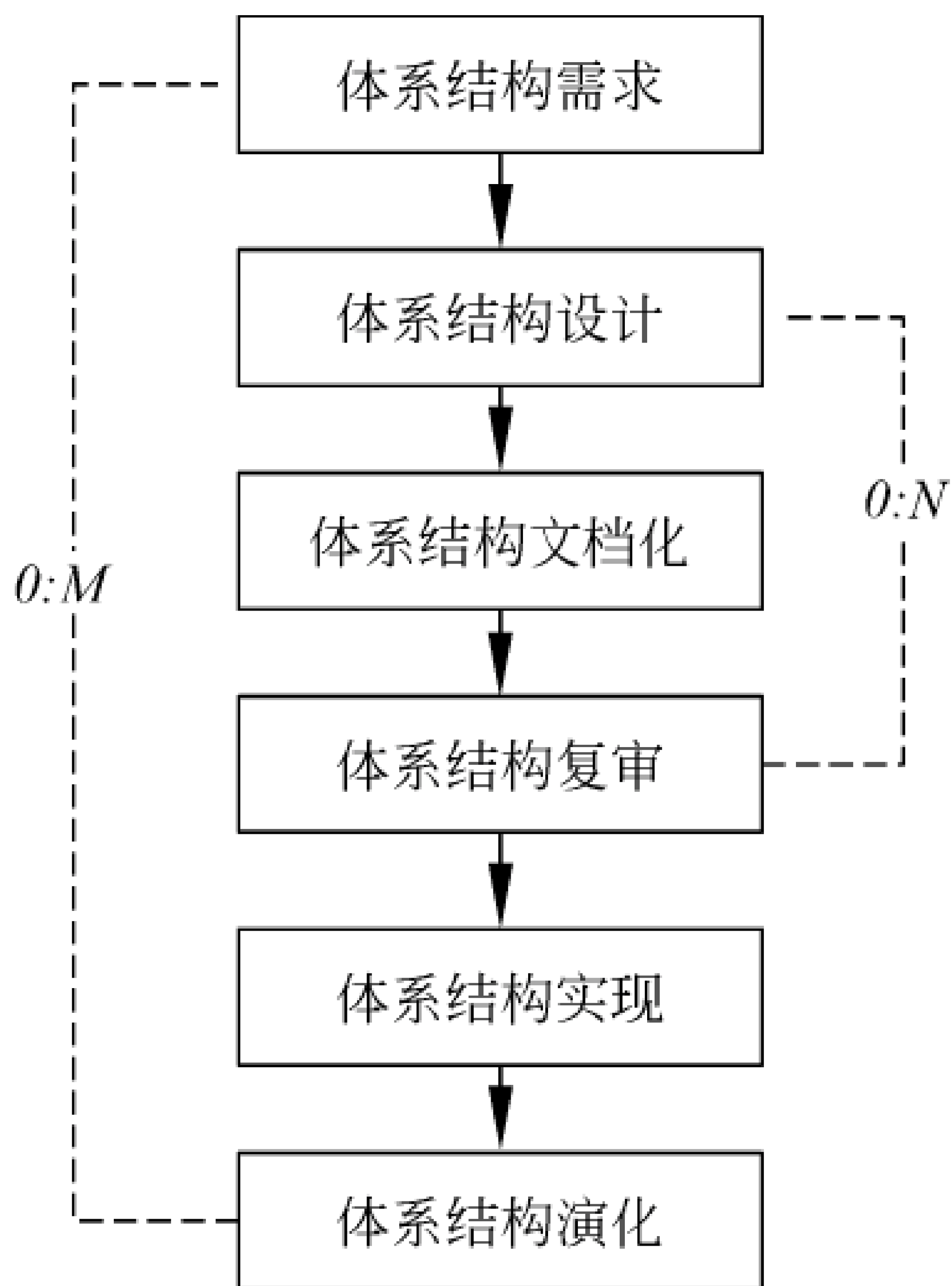


图 5-2　体系结构开发模型

5.2.4　体系结构需求

需求是指用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。体系结构需求受技术环境和体系结构设计师的经验影响。需求过程主要是获取用户需求，标识系统中所要用到的构件。体系结构需求过程如图 5-3 所示。如果以前有类似的系统体系结构的需求，我们可以从需求库中取出，加以利用和修改，以节省需求获取的时间，减少重复劳动，提高开发效率。

1．需求获取

体系结构需求一般来自三个方面，分别是系统的质量目标、系统的商业目标和系统开发人员的商业目标。软件体系结构需求获取过程主要是定义开发人员必须实现的软件功能，使得用户能完成他们的任务，从而满足业务上的功能需求。与此同时，还要获得软件质量属性，满足一些非功能需求。

2．标识构件

在图 5-3 中虚框部分属于标识构件过程，该过程为系统生成初始逻辑结构，包含大致的构件。这一过程又可分为三步来实现。

第一步：生成类图。生成类图的 CASE 工具有很多，例如 Rational Rose 2000 能自动生成类图。

第二步：对类进行分组。在生成的类图基础上，使用一些标准对类进行分组可以大大简化类图结构，使之更清晰。一般地，与其他类隔离的类形成一个组，由概括关联的类组成一个附加组，由聚合或合成关联的类也

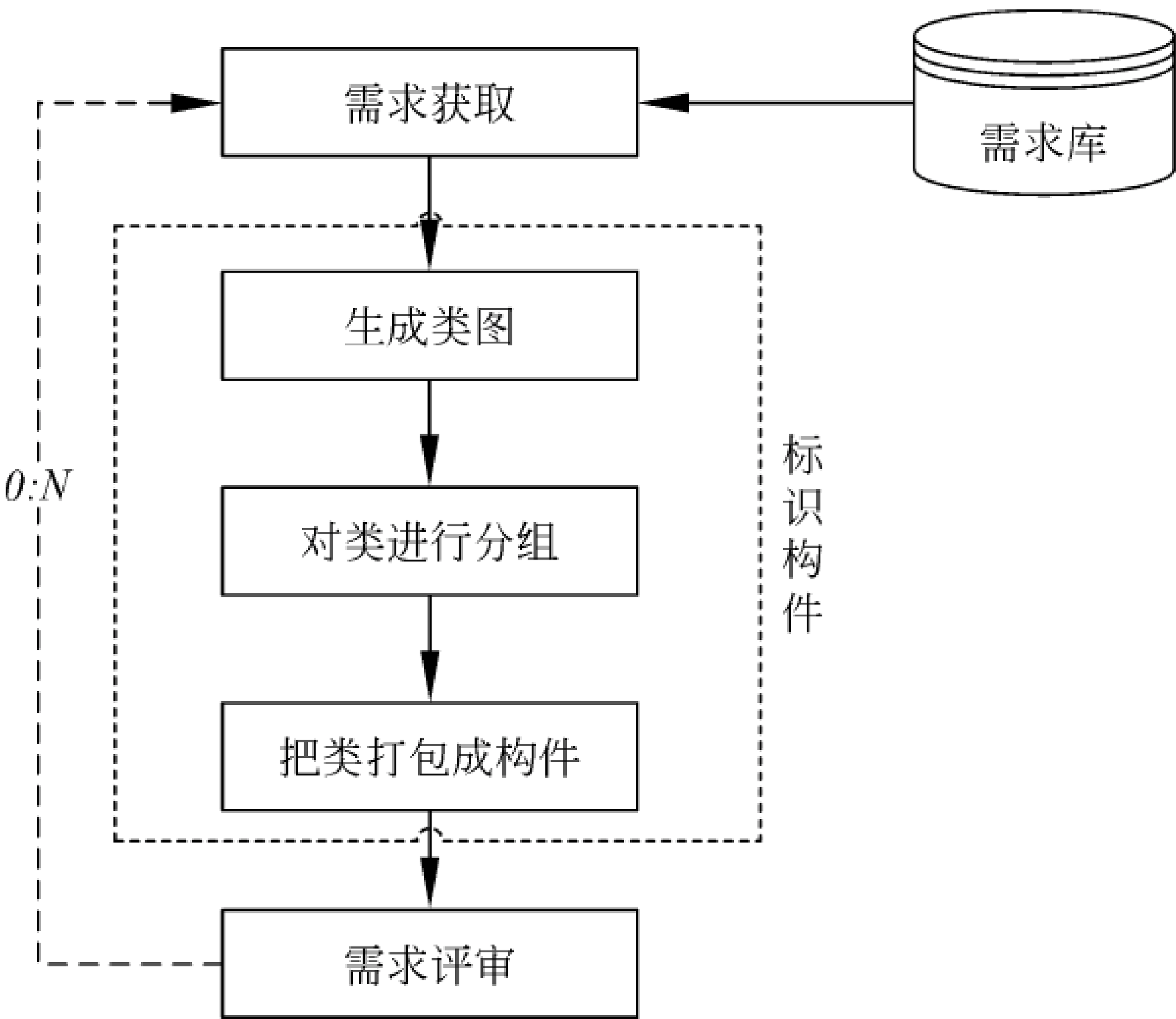


图 5-3　体系结构需求过程

形成一个附加组。

第三步：把类打包成构件。把在第二步得到的类簇打包成构件，这些构件可以分组合并成更大的构件。

3．架构需求评审

组织一个由不同代表（如分析人员、客户、设计人员、测试人员）组成的小组，对体系结构需求及相关构件进行仔细的审查。审查的主要内容包括所获取的需求是否真实反映了用户的要求，类的分组是否合理，构件合并是否合理等。必要时，可以在“需求获取—标识构件—需求评审”之间进行迭代。

5.2.5 体系结构设计

体系结构需求用来激发和调整设计决策，不同的视图被用来表达与质量目标有关的信息。体系结构设计是一个迭代过程，如果要开发的系统能够从已有的系统中导出大部分，则可以使用已有系统的设计过程。软件体系设计过程如图 5-4 所示。

1．提出软件体系结构模型

在建立体系结构的初期，选择一个合适的体系结构风格是首要的。在这个风格基础上，开发人员通过体系结构模型，可以获得关于体系结构属性的理解。此时，虽然这个模型是理想化的（其中的某些部分可能错误地表示了应用的特征），但是，该模型为将来的实现和演化过程建立了目标。



图 5-4 体系结构设计过程

2．把已标识的构件映射到软件体系结构中

把在体系结构需求阶段已标识的构件映射到体系结构中，将产生一个中间结构，这个中间结构只包含那些能明确适合体系结构模型的构件。

3．分析构件之间的相互作用

为了把所有已标识的构件集成到体系结构中，必须认真分析这些构件的相互作用和关系。

4．产生软件体系结构

一旦决定了关键的构件之间的关系和相互作用，就可以在第 2 阶段得到的中间结构的基础上进行精化。

5．设计评审

一旦设计了软件体系结构，必须邀请独立于系统开发的外部人员对体系结构进行评审。

5.2.6　体系结构文档化

绝大多数的体系结构都是抽象的，由一些概念上的构件组成。例如，层的概念在任何程序设计语言中都不存在。因此，要让系统分析员和程序员去实现体系结构，还必须得把体系结构进行文档化。文档是在系统演化的每一个阶段，系统设计与开发人员的通信媒介，是为验证体系结构设计和提炼或修改这些设计（必要时）所执行预先分析的基础。

体系结构文档化过程的主要输出结果是体系结构规格说明和测试体系结构需求的质量设计说明书这两个文档。生成需求模型构件的精确的形式化的描述，作为用户和开发者之间的一个协约。软件体系结构的文档要求与软件开发项目中的其他文档是类似的。文档的完整性和质量是软件体系结构成功的关键因素。文档要从使用者的角度进行编写，必须分发给所有与系统有关的开发人员，且必须保证开发者手上的文档是最新的。

5.2.7　体系结构复审

从图 5-2 中可以看出，体系结构设计、文档化和复审是一个迭代过程。从这个方面来说，在一个主版本的软件体系结构分析之后，要安排一次由外部人员（用户代表和领域专家）参加的复审。

鉴于体系结构文档标准化，以及风险识别的现实情况，通常我们根据架构设计，搭建一个可运行的最小化系统用于评估和测试体系架构是否满足需要。是否存在可识别的技术和协作风险。

复审的目的是标识潜在的风险，及早发现体系结构设计中的缺陷和错误，包括体系结构能否满足需求、质量需求是否在设计中得到体现、层次是否清晰、构件的划分是否合理、文档表达是否明确、构件的设计是否满足功能与性能的要求等。

5.2.8　体系结构实现

所谓“实现”就是要用实体来显示出一个软件体系结构，即要符合体系结构所描述的结构性设计决策，分割成规定的构件，按规定方式互相交互。体系结构的实现过程如图 5-5 所示。

图 5-5 中的虚框部分是体系结构的实现过程。整个实现过程是以复审后的文档化的体系

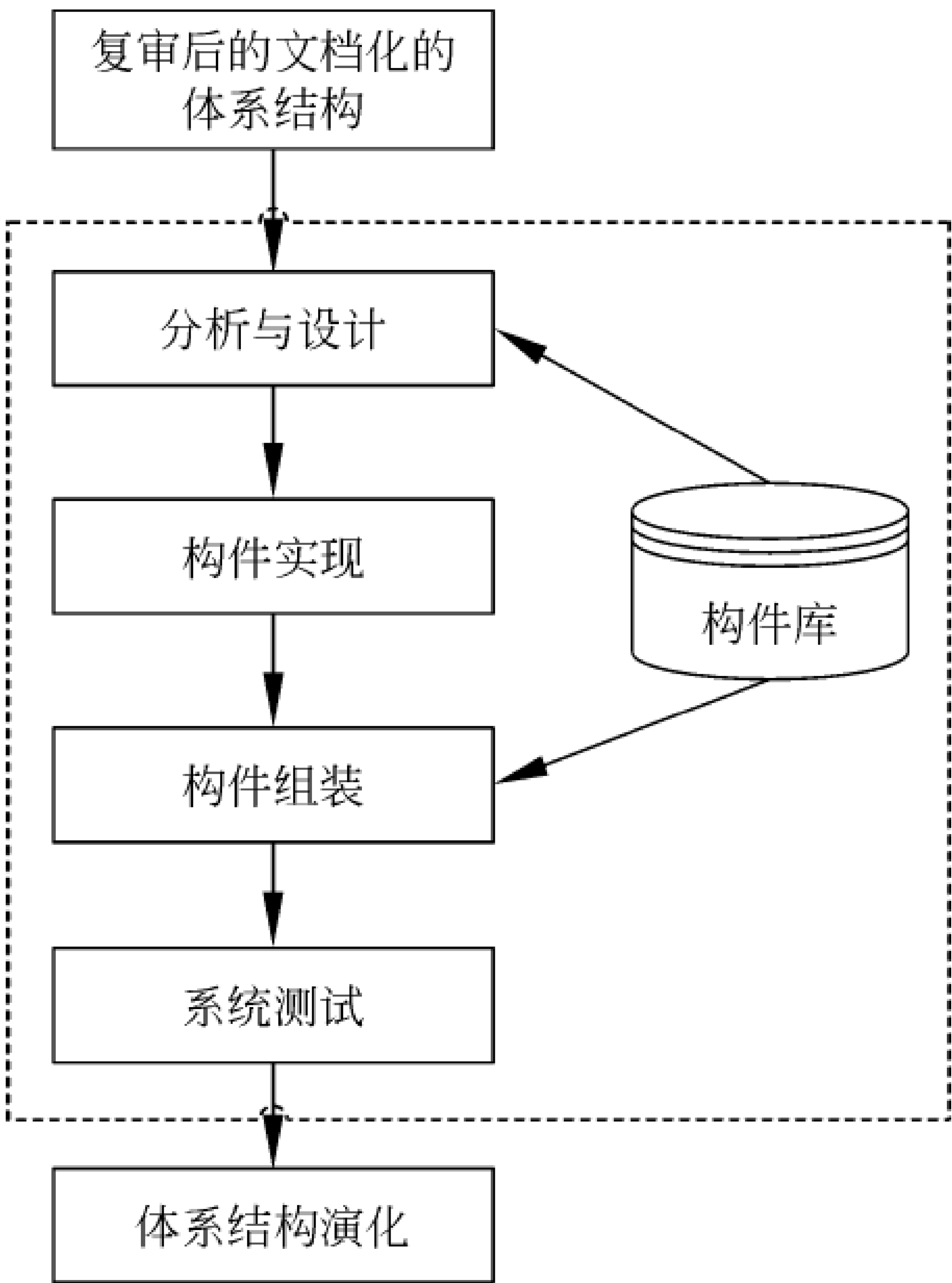


图 5-5 体系结构实现过程

结构说明书为基础的，每个构件必须满足软件体系结构中说明的对其他构件的责任。这些决定即实现的约束是在系统级或项目范围内给出的，每个构件上工作的实现者是看不见的。

在体系结构说明书中，已经定义了系统中的构件与构件之间的关系。因为在体系结构层次上，构件接口约束对外唯一地代表了构件，所以可以从构件库中查找符合接口约束的构件，必要时开发新的满足要求的构件。然后，按照设计提供的结构，通过组装支持工具把这些构件的实现体组装起来，完成整个软件系统的连接与合成。

最后一步是测试，包括单个构件的功能性测试和被组装应用的整体功能和性能测试。

5.2.9 体系结构的演化

在构件开发过程中，用户的需求可能还有变动。在软件开发完毕，正常运行后，由一个单位移植到另一个单位，需求也会发生变化。在这两种情况下，就必须相应地修改软件体系结构，以适应新的变化了的软件需求。体系结构演化过程如图 5-6 所示。

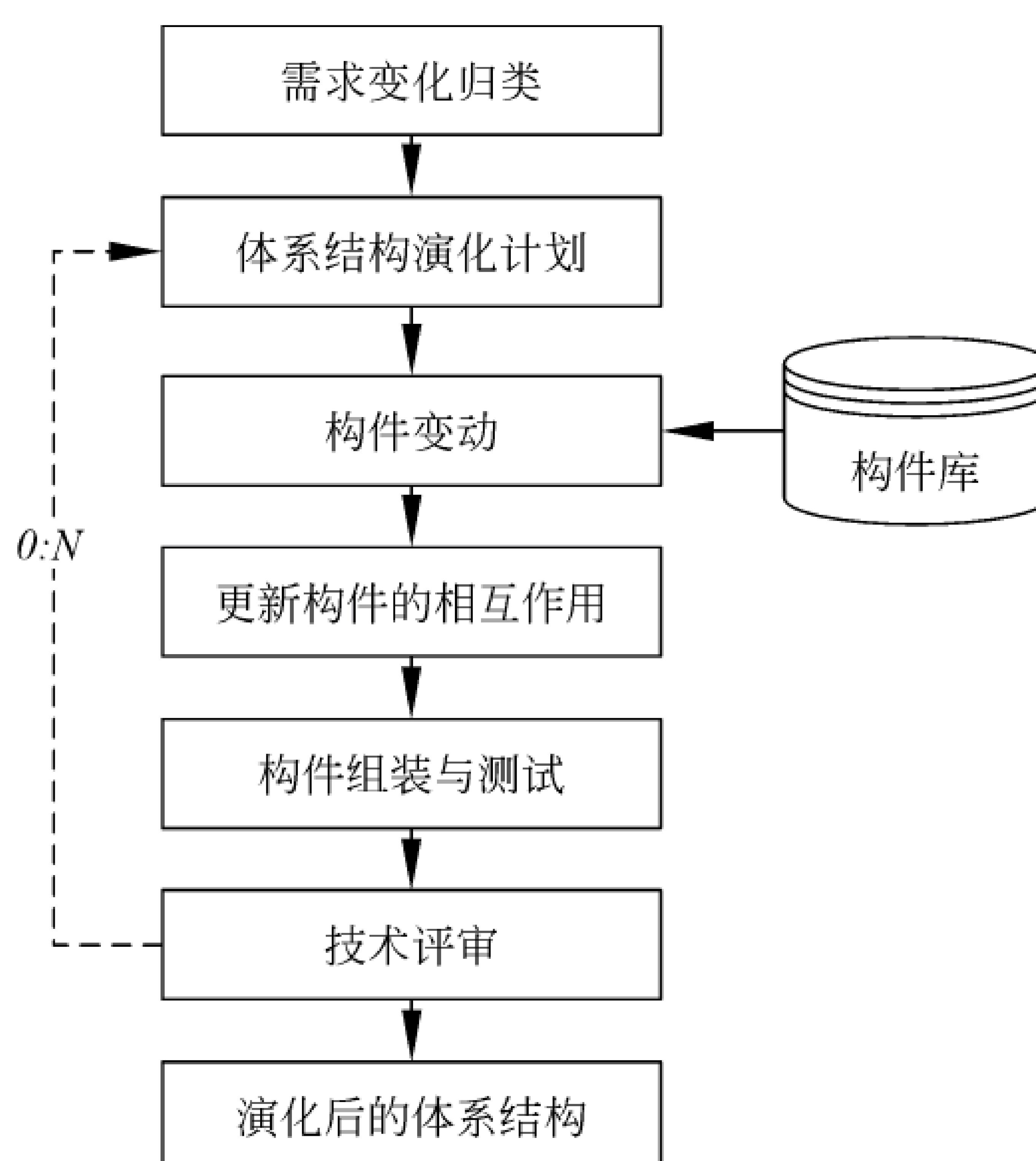


图 5-6 体系结构演化过程

体系结构演化是使用系统演化步骤去修改应用，以满足新的需求。主要包括以下 6 个步骤。

1．需求变化归类

首先必须对用户需求的变化进行归类，使变化的需求与已有构件对应。对找不到对应构件的变动，也要做好标记，在后续工作中，将创建新的构件，以对应这部分变化的需求。

2．制订体系结构演化计划

在改变原有结构之前，开发组织必须制订一个周密的体系结构演化计划，作为后续演化开发工作的指南。

3．修改、增加或删除构件

在演化计划的基础上，开发人员可根据在第 1 步得到的需求变动的归类情况，决定是否修改或删除存在的构件、增加新构件。最后，对修改和增加的构件进行功能性测试。

4．更新构件的相互作用

随着构件的增加、删除和修改，构件之间的控制流必须得到更新。

5．构件组装与测试

通过组装支持工具把这些构件的实现体组装起来，完成整个软件系统的连接与合

成，形成新的体系结构。然后对组装后的系统整体功能和性能进行测试。

6．技术评审

对以上步骤进行确认，进行技术评审。评审组装后的体系结构是否反映需求变动，符合用户需求。如果不符合，则需要在第 2 到第 6 步之间进行迭代。

在原来系统上所作的所有修改必须集成到原来的体系结构中，完成一次演化过程。

5.3　软件架构风格

软件体系结构设计的一个核心目标是重复的体系结构模式，即达到体系结构级的软件重用。也就是说，在不同的软件系统中，使用同一体系结构。基于这个目的，主要任务是研究和实践软件体系结构的风格和类型问题。

5.3.1　软件架构风格概述

软件体系结构风格是描述某一特定应用领域中系统组织方式的惯用模式。体系结构风格定义一个系统家族，即一个体系结构定义一个词汇表和一组约束。词汇表中包含一些构件和连接件类型，而这组约束指出系统是如何将这些构件和连接件组合起来的。体系结构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统。对软件体系结构风格的研究和实践促进对设计的重用，一些经过实践证实的解决方案也可以可靠地用于解决新的问题。例如，如果某人把系统描述为“客户/服务器”模式，则不必给出设计细节，我们立刻就会明白系统是如何组织和工作的。

5.3.2　经典软件体系结构风格

1．管道和过滤器

在管道/过滤器风格的软件体系结构（见图 5-7）中，每个构件都有一组输入和输出，数据输入构件，经过内部处理，然后产生数据输出。因此，这里的构件被称为过滤器，这种风格的连接件就像是数据流传输的管道，将一个过滤器的输出传到另一过滤器的输入。

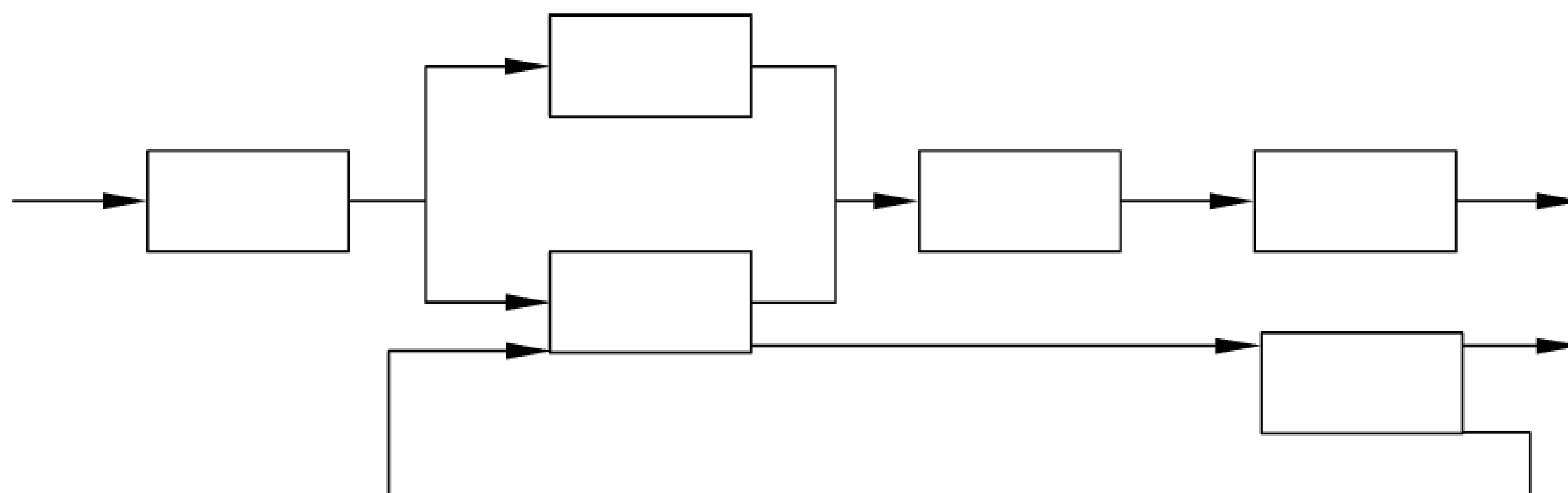


图 5-7　管道/过滤器风格的体系结构

2．数据抽象和面向对象组织

抽象数据类型概念对软件系统有着重要作用，目前软件界已普遍转向使用面向对象系统。这种风格建立在数据抽象和面向对象的基础上，数据的表示方法和它们的相应操作封装在一个抽象数据类型或对象中。这种风格的构件是对象，或者说是抽象数据类型的实例（见图 5-8）。

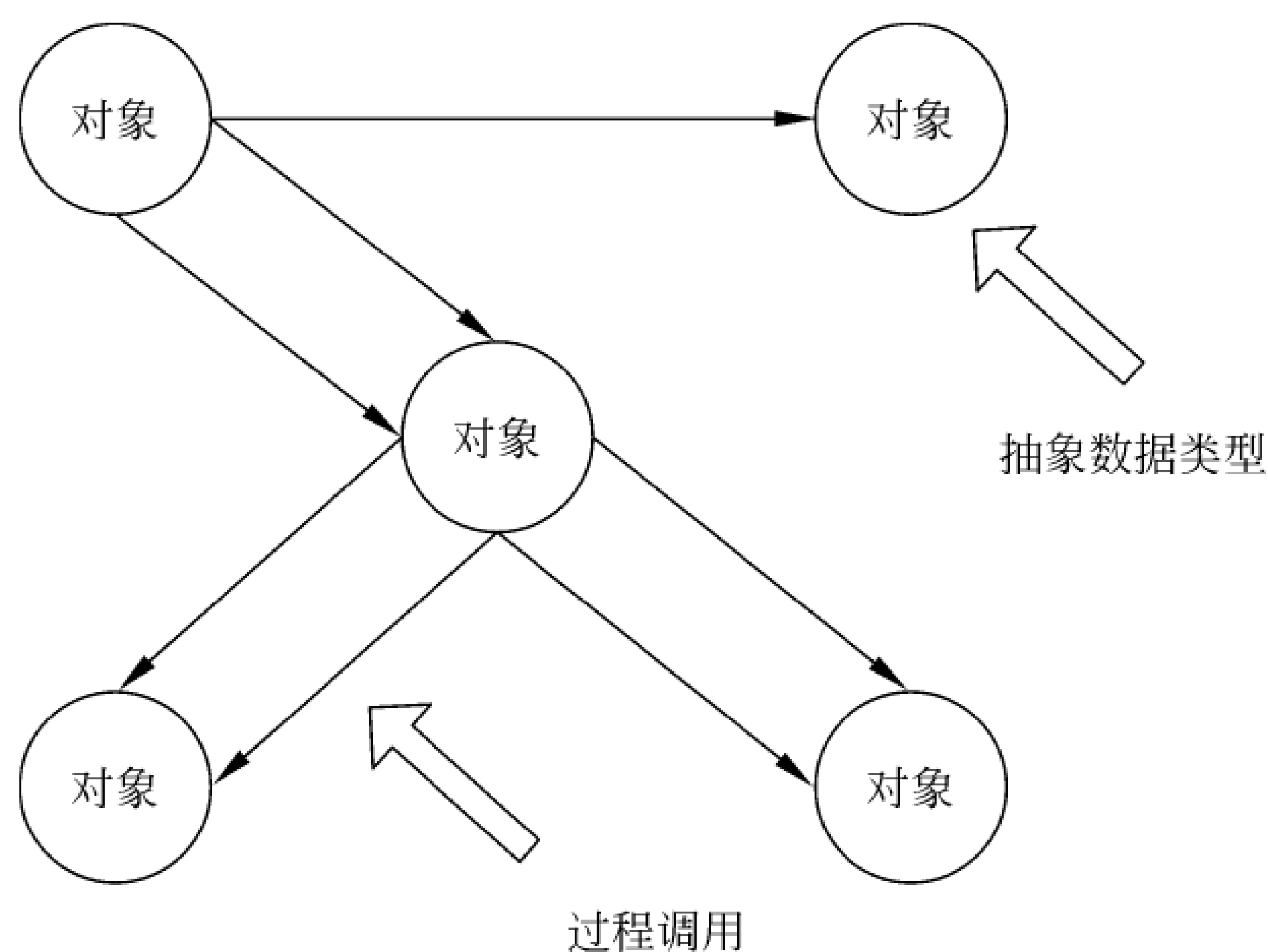


图 5-8 数据抽象和面向对象风格的体系结构

3．事件驱动系统

事件驱动系统风格是构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其他构件中的过程在一个或多个事件中注册。当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。

4．分层系统

层次系统（见图 5-9）组成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层接口只对相邻的层可见。这样的系统中构件在层上实现了虚拟机。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件重用提供了强大的支持。

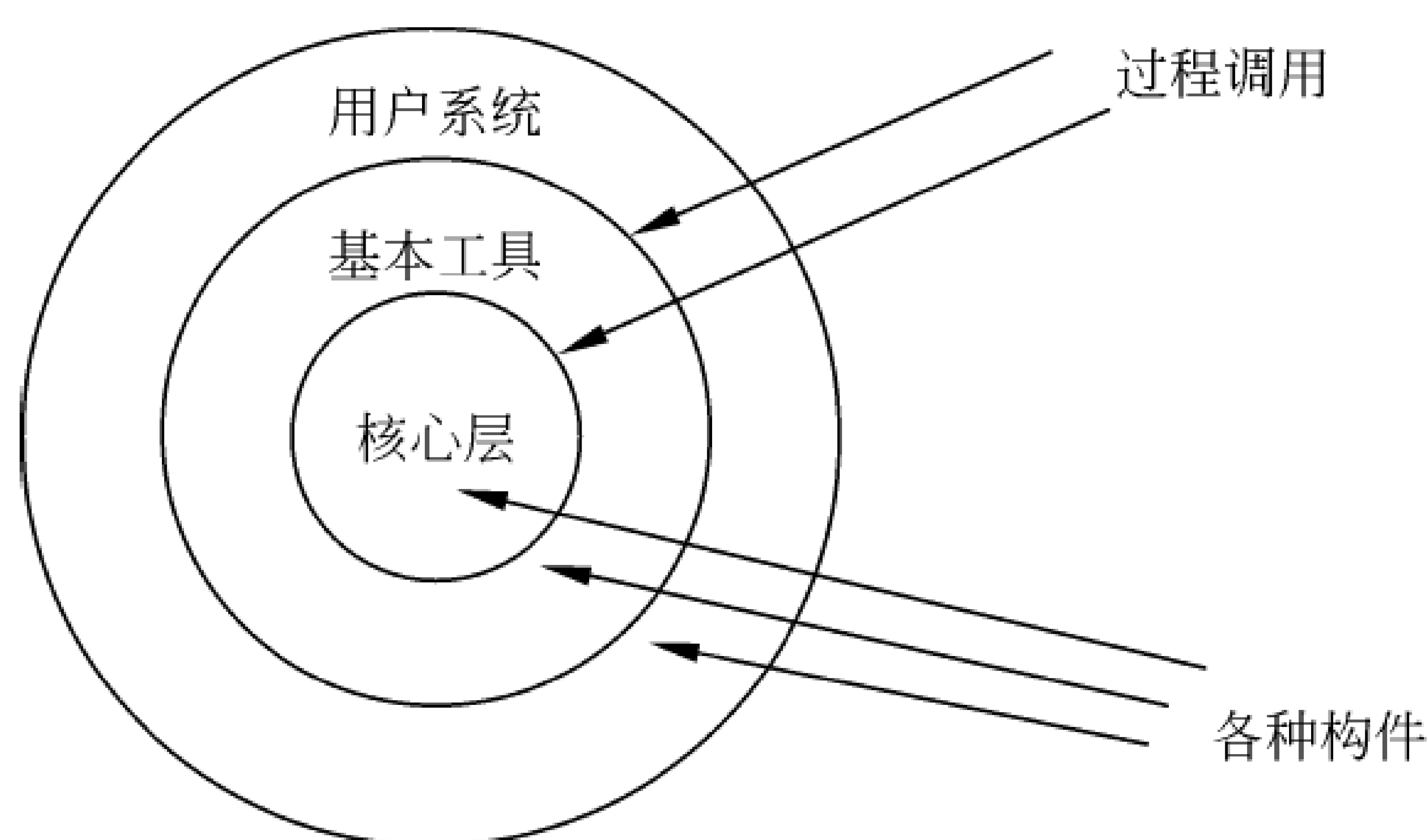


图 5-9 层次系统风格示意图

5．仓库系统及知识库

在仓库（repository）风格中，有两种不同的构件：中央数据结构说明当前状态，独

立构件在中央数据存储上执行。

一方面，若构件控制共享数据，则仓库是一传统型数据库；另一方面，若中央数据结构的当前状态触发进程执行的选择，则仓库是一黑板系统（见图 5-10）。

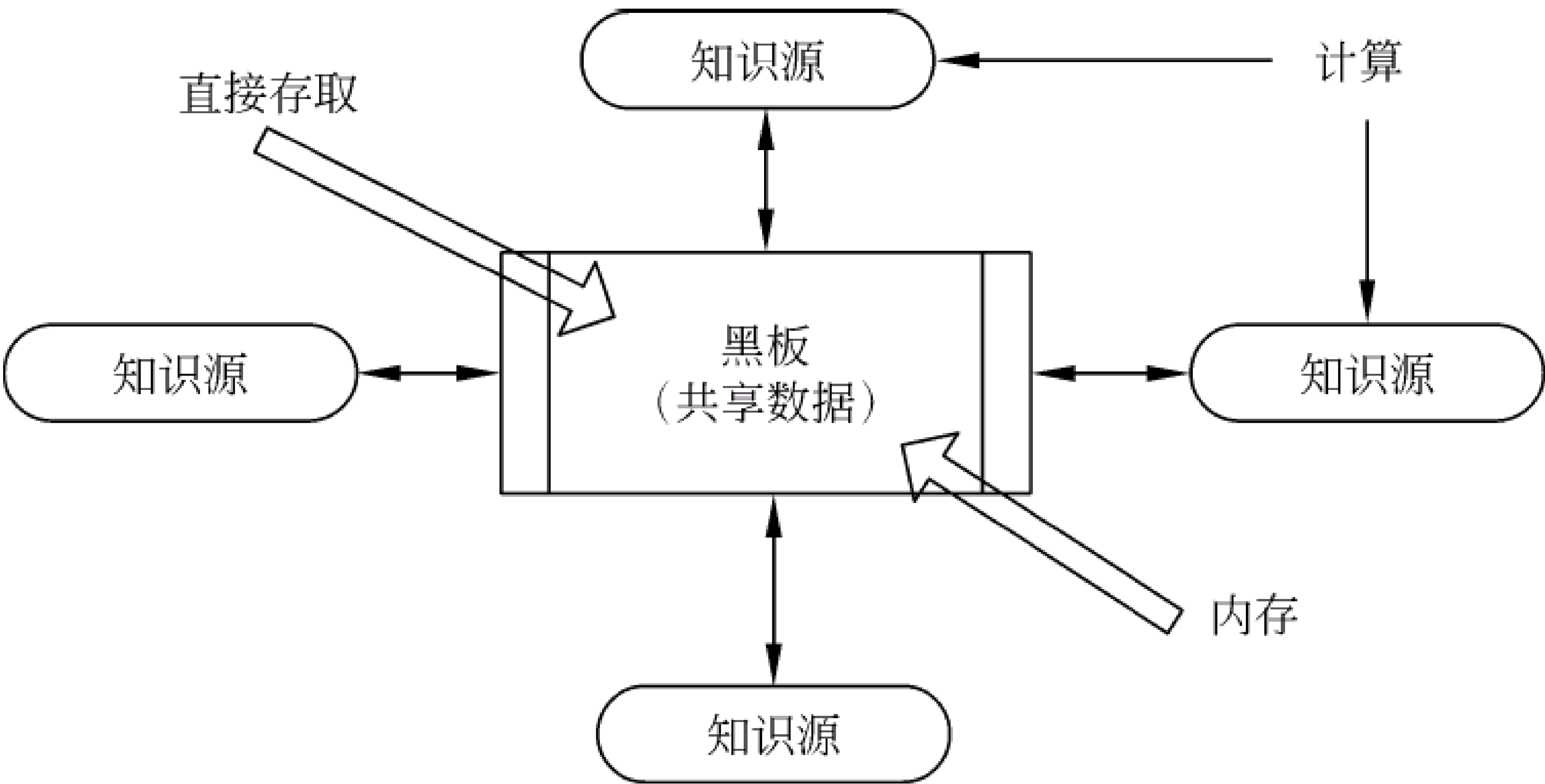


图 5-10　黑板系统的组成

6．C2 风格

C2 体系结构风格可以概括为通过连接件绑定在一起按照一组规则运作的并行构件网络。C2 风格中的系统组织规则如下。

（1）系统中的构件和连接件都有一个顶部和一个底部。

（2）构件的顶部应连接到某连接件的底部，构件的底部则应连接到某连接件的顶部。而构件与构件之间的直接连接是不允许的。

（3）一个连接件可以和任意数目的其他构件和连接件连接。

（4）当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部。

C2 风格如图 5-11 所示。图中构件与连接件之间的连接体现了 C2 风格中构建系统的规则。

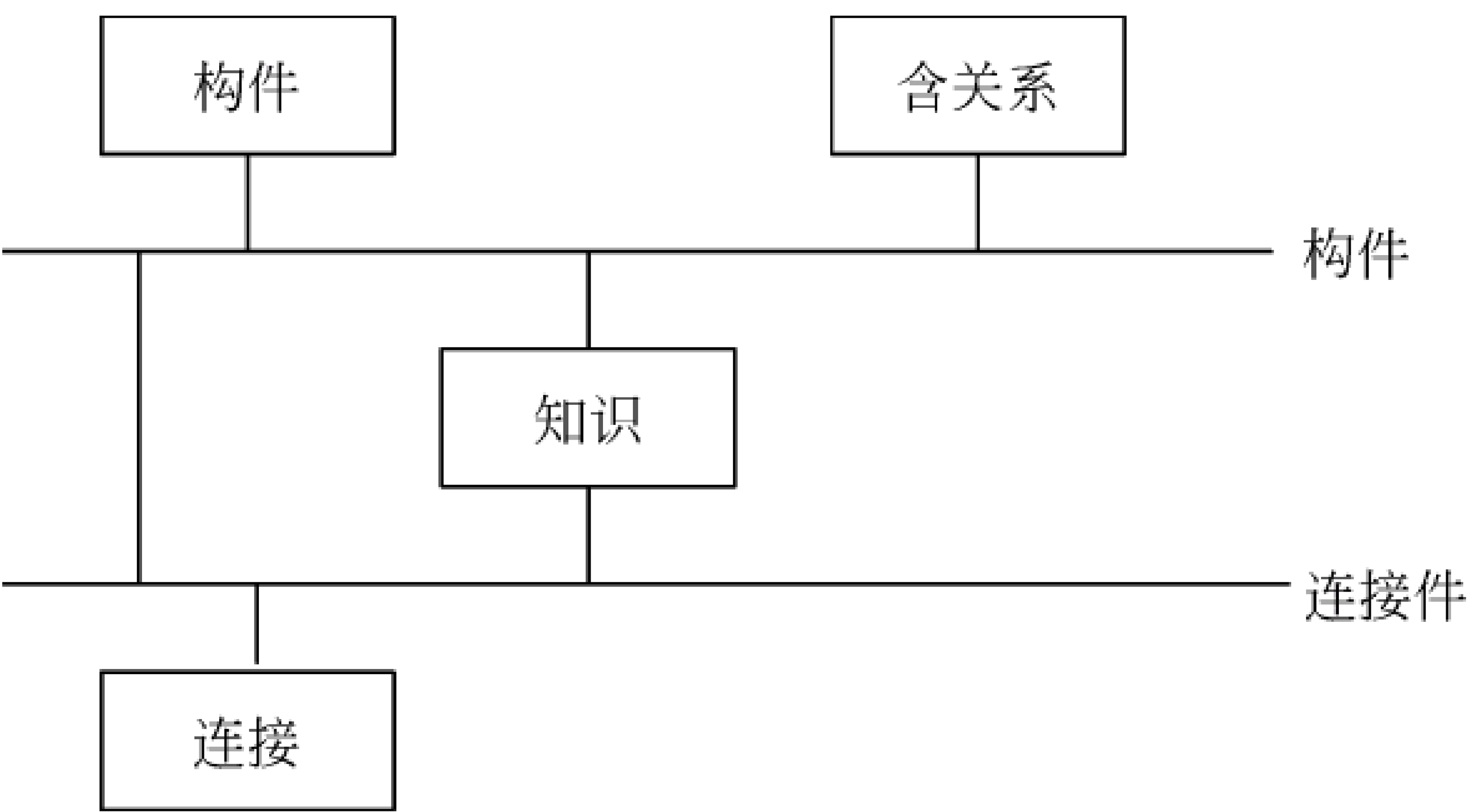


图 5-11　C2 风格的体系结构

5.3.3 客户/服务器风格

客户/服务器（C/S）计算技术在信息产业中占有重要的地位。网络计算经历了从基于宿主机的计算模型到客户/服务器计算模型的演变。在集中式计算技术时代，广泛使用的是大型机/小型机计算模型。它是通过一台物理上与宿主机相连接的非智能终端来实现宿主机上的应用程序。在多用户环境中，宿主机应用程序即负责与用户的交互，又负责对数据的管理。宿主机上的应用程序一般也分为与用户交互的前端和管理数据的后端，即数据库管理系统（DBMS）集中式的系统使用户能共享贵重的硬件设备。如磁盘机、打印机和调制解调器等。

C/S 软件体系结构是基于资源不对等且实现共享而提出的，是在 20 世纪 90 年代成熟的技术，C/S 体系结构定义了工作站如何与服务器相连，实现部分数据和应用分布到多个处理机上。C/S 体系结构有三个主要组成部分：数据库服务器、客户应用程序和网络，如图 5-12 所示。

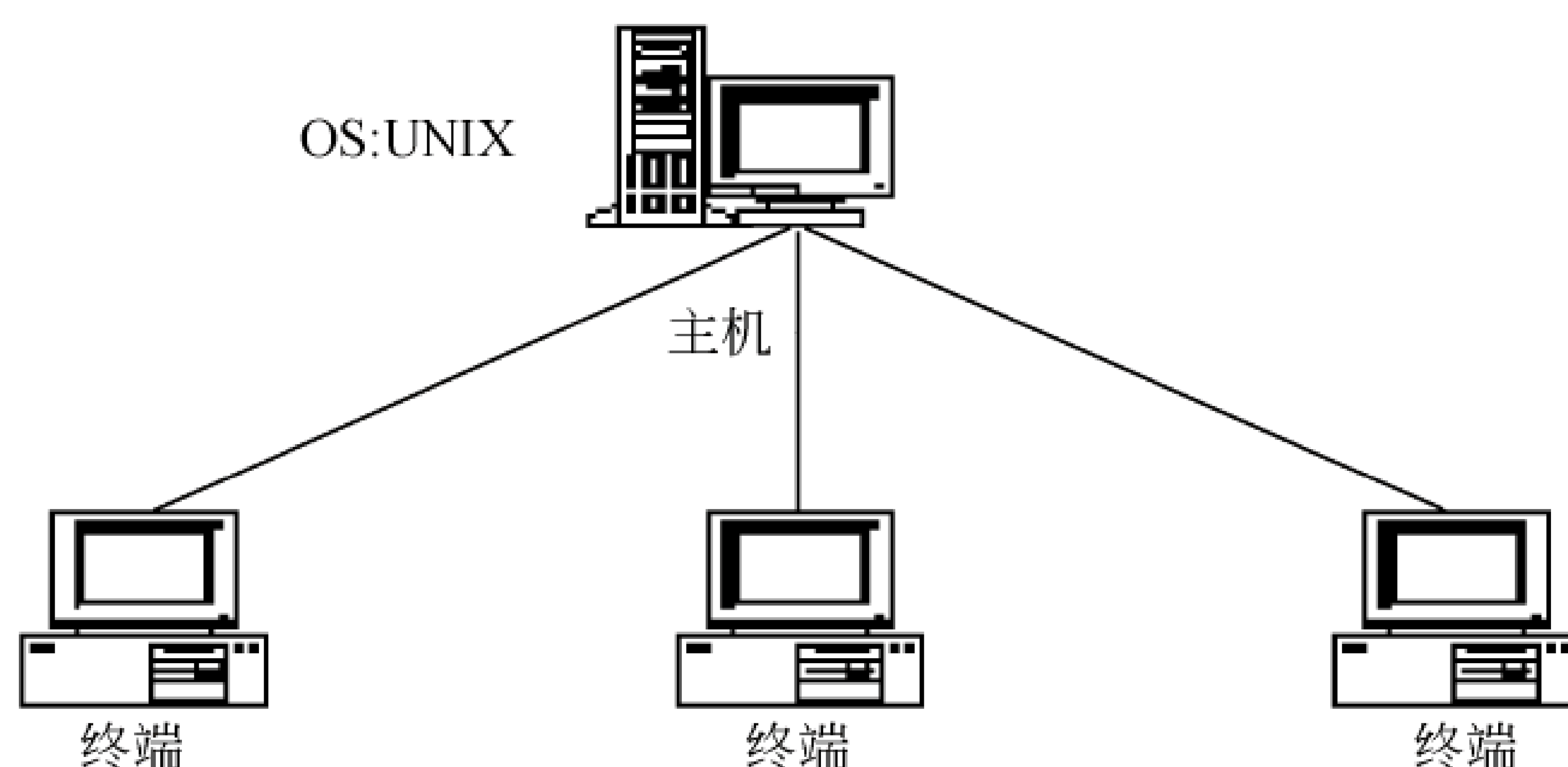


图 5-12　C/S 体系结构示意图

服务器负责有效地管理系统的资源，例如，数据库管理系统，其任务集中于：

（1）数据库安全性的要求。

（2）数据库访问并发性的控制。

（3）数据库前端的客户应用程序的全局数据完整性规则。

（4）数据库的备份与恢复。

客户应用程序的主要任务如下。

（1）提供用户与数据库交互的界面。

（2）向数据库服务器提交用户请求并接收来自数据库服务器的信息。

（3）利用客户应用程序对存在于客户端的数据执行应用逻辑要求。

C/S 体系结构的优点主要在于系统的客户应用程序和服务器构件分别运行在不同的计算机上，系统中每台服务器都可以适合各构件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小。在 C/S 体系结构中，系统中的功能构件充分隔离，客户应用程序的开发集中于数据的显示和分析，而数据库服务器的开发则集中于数据的管理，不必在每一个新的应用程序中都要对一个 DBMS 进行编码。将大应用处理任务分布到许多通过网络连接的低成本计算机上，以节约大量费用。

C/S 体系结构具有强大的数据操作和事务处理能力，模型思想简单，易于人们理解和接受。但随着企业规模的日益扩大，软件的复杂程度不断提高，C/S 体系结构逐渐暴

露了以下缺点。

（1）开发成本较高。C/S 体系结构对客户端软硬件配置要求较高，尤其是软件的不断升级，对硬件要求不断提高，增加了整个系统的成本，且客户端变得越来越臃肿。

（2）客户端程序设计复杂。采用 C/S 体系结构进行软件开发，大部分工作量放在客户端的程序设计上，客户端显得十分庞大。

（3）信息内容和形式单一，因为传统应用一般为事务处理，界面基本遵循数据库的字段解释，开发之初就已确定，而且不能随时截取办公信息和档案等外部信息，用户获得的只是单纯的字符和数字，既枯燥又死板。

（4）用户界面风格不一，使用繁杂，不利于推广使用。

（5）软件移植困难。采用不同开发工具或平台开发的软件一般互不兼容，不能或很难移植到其他平台上运行。

（6）软件维护和升级困难。采用 C/S 体系结构的软件要升级，开发人员必须到现场为客户机升级，每个客户机上的软件都需维护。对软件的一个小小改动（例如只改动一个变量），每一个客户端都必须更新。

5.3.4　三层 C/S 结构风格

针对二层 C/S 体系结构的缺点，三层 C/S 体系结构应运而生。其结构如图 5-13 所示。在三层 C/S 体系结构中，增加了一个应用服务器。可以将整个应用逻辑驻留在应用服务器上，而只有表示层存在于客户机上。这种结构被称为“瘦客户机”。三层 C/S 体系结构是将应用功能分成表示层、功能层和数据层三个部分。

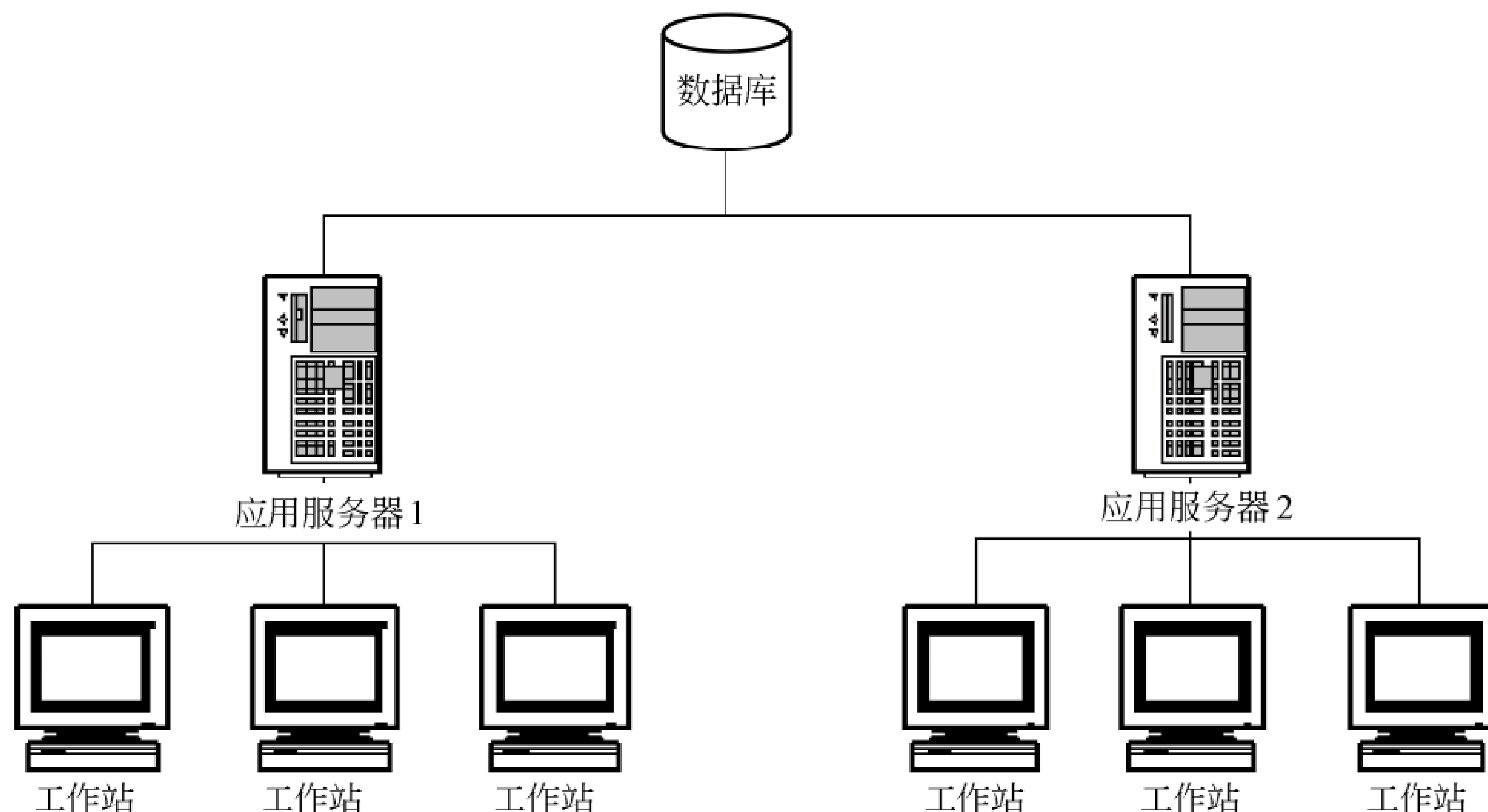


图 5-13　三层 C/S 结构示意图

1）表示层

表示层是应用的用户接口部分担负与应用逻辑间的对话功能。它用于用户从工作站输入的数据，并显示应用输出的数据。为使用户能直观地进行操作，一般要使用图形用户界面（Graphic User Interface，GUI），在变更用户界面时，只需改写显示控制和数据检查程序，而不影响业务逻辑。

2）功能层

功能层是应用的本体，它负责具体的业务处理逻辑，例如在制作订购合同时要计算合同金额。表示层和功能层之间的数据互交要尽可能简洁。例如，用户检索数据时，要将有关检索要求的信息一次性地传送给功能层，检索结果数据也由功能层一次性地传送给表示层。

3）数据层

数据层通常是数据库管理系统，负责管理对数据库数据的读写。数据库系统必须能迅速执行大量数据的更新和检索。

三层 C/S 的解决方案对这三层进行明确分割，不同层构件相互独立，层间的接口简洁，适合复杂事务处理。

5.3.5 浏览器/服务器风格

浏览器/服务器（browser/server，B/S）风格就是上述三层应用结构的一种实现方式。其具体结构为浏览器/Web 服务器/数据库服务器。三层 C/S 的解决方案相比，客户端采用 WWW 浏览器，应用服务器是 Web 服务器。B/S 体系结构主要是利用不断成熟的 WWW 浏览器技术，结合浏览器的多种脚本语言，用通用浏览器就实现了原来需要复杂的专用软件才能实现的强大功能，并节约了开发成本。从某种程度上来说 B/S 结构是种全新的软件体系结构。

在 B/S 结构中，除了数据库服务器外，应用程序以网页形式存放于 Web 服务器上，用户运行某个应用程序时只需在客户端上的浏览器中键入相应的网址（URL），调用 Web 服务器上的应用程序并对数据库进行操作完成相应的数据处理工作，最后将结果通过浏览器显示给用户。

基于 B/S 体系结构的软件，系统安装、修改和维护全在服务器端解决。用户在使用系统时，仅仅需要一个浏览器就可运行全部的模块。真正达到了“零客户端”的功能，很容易在运行时自动升级。B/S 体系结构还提供了异种机、异种网、异种应用服务的联机、联网等。

与 C/S 体系结构相比，B/S 体系结构也有许多不足之处，例如：

（1）B/S 体系结构缺乏对动态页面的支持能力，没有集成有效的数据库处理功能。

（2）B/S 体系结构的系统扩展能力差，安全性较难以控制。

（3）采用 B/S 体系结构的应用系统，在数据查询等响应速度上，要远远地低于 C/S

体系结构。

（4）BS 体系结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理（online transaction processing，OLTP）应用。

因此，虽然 B/S 结构的计算机应用系统有如此多的优越性，但由于 C/S 结构的成熟性且 C/S 结构的计算机应用系统网络负载较小，因此，应用系统常以 C/S 和 B/S 混合应用形式出现，如图 5-14 所示。

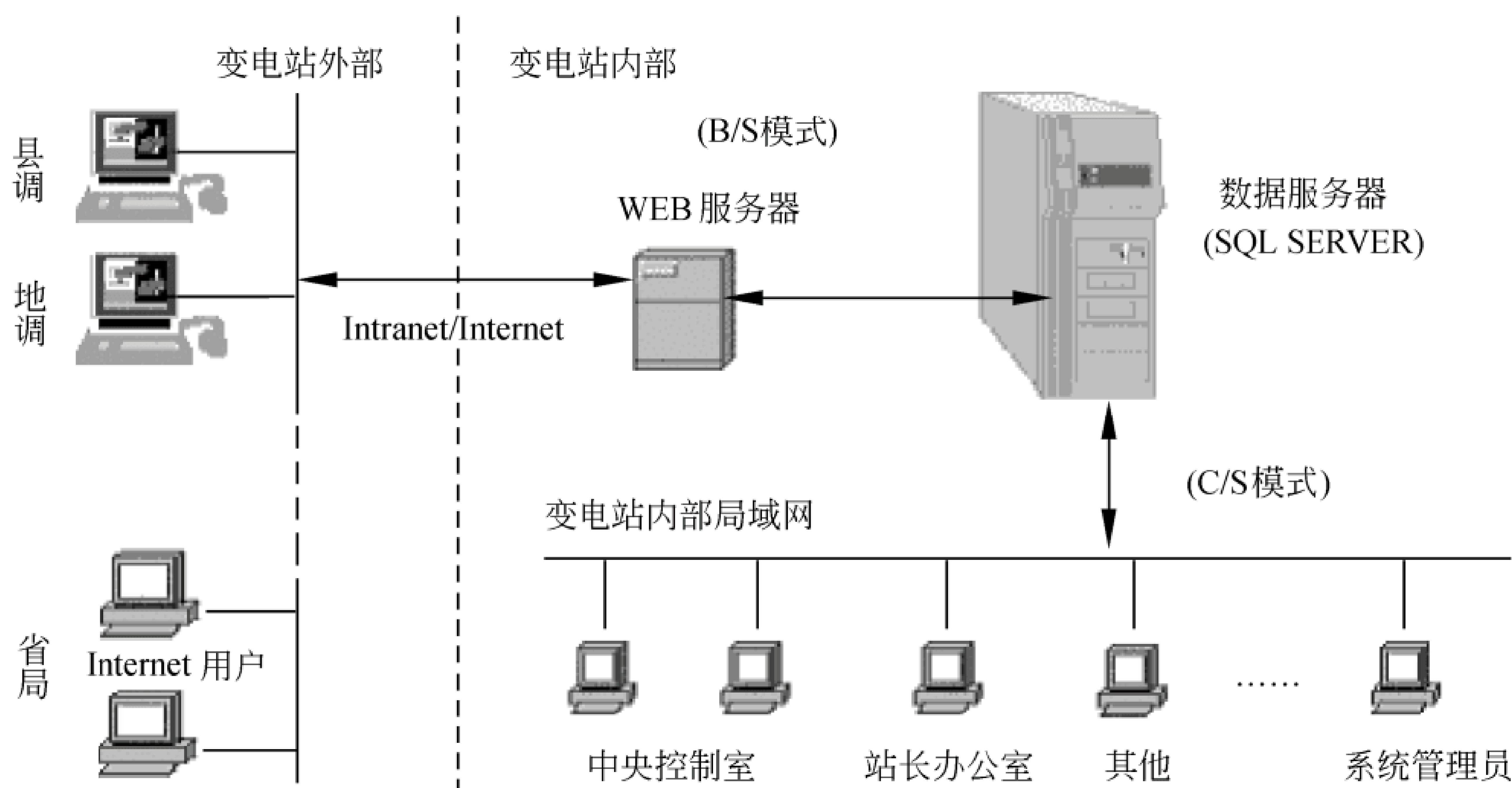


图 5-14　C/S 与 B/S 混合体系结构风格

上图描述了供电调度系统的结构，内部采用 C/S 风格，对外采用 B/S 风格，它针对不同应用和客户需要，充分利用了两种体系结构的优点。

5.4　特定领域软件体系结构

早在 20 世纪 70 年代就有人提出程序族、应用族的概念，特定领域软件体系结构的主要目的是在一组相关的应用中共享软件体系结构。

5.4.1　DSSA 的定义

简单地说，（Domain Specific Software Architecture，DSSA）就是在一个特定应用领域中为一组应用提供组织结构参考的标准软件体系结构。对 DSSA 研究的角度、关心的问题不同导致了对 DSSA 的不同定义。

Hayes Roth 对 DSSA 的定义如下：“DSSA 就是专用于一类特定类型的任务（领域）的、在整个领域中能有效地使用的、为成功构造应用系统限定了标准的组合结构的软件构件的集合。”

Tracz 的定义为："DSSA 就是一个特定的问题领域中支持一组应用的领域模型、参考需求、参考体系结构等组成的开发基础，其目标就是支持在一个特定领域中多个应用的生成。"

通过对众多的 DSSA 的定义和描述的分析，可知 DSSA 的必备特征如下：

（1）一个严格定义的问题域和问题解域。

（2）具有普遍性。使其可以用于领域中某个特定应用的开发。

（3）对整个领域的构件组织模型的恰当抽象。

（4）具备该领域固定的、典型的在开发过程中可重用元素。

一般的 DSSA 的定义并没有对领域的确定和划分给出明确说明。从功能覆盖的范围角度有两种理解 DSSA 中领域的含义的方式。

（1）垂直域：定义了一个特定的系统族，包含整个系统族内的多个系统，结果是在该领域中可作为系统的可行解决方案的一个通用软件体系结构。

（2）水平域：定义了在多个系统和多个系统族中功能区域的共有部分。在子系统级上涵盖多个系统族的特定部分功能。

在垂直域上定义的 DSSA 只能应用于一个成熟的、稳定的领域，但这个条件比较难以满足：若将领域分割成较小的范围，则更相对容易，也容易得到一个一致的解决方案。

5.4.2 DSSA 的基本活动

实施 DSSA 的过程中包含了一些基本的活动。虽然具体的 DSSA 方法可能定义不同的概念、步骤和产品等，但这些基本活动大体上是一致的。以下将分三个阶段介绍这些活动。

1．领域分析

这个阶段的主要目标是获得领域模型。领域模型描述领域中系统之间的共同的需求，即领域模型所描述的需求为领域需求。在这个阶段中首先要进行一些准备性的活动，包括定义领域的边界。从而明确分析的对象；识别信息源，整个领域工程过程中信息的来源，可能的信息源包括现存系统、技术文献、问题域和系统开发的专家、用户调查和市场分析、领域演化的历史记录等，在此基础上就可以分析领域中系统的需求，确定哪些需求是领域中的系统广泛共享的，从而建立领域模型。当领域中存在大量系统时，需要选择它们的一个子集作为样本系统。对样本系统需求的考察将显示领域需求的一个变化范围。一些需求对所有被考察的系统是共同的，一些需求是单个系统所独有的。很多需求位于这两个极端之间，即被部分系统共享。

2．领域设计

这个阶段的目标是获得 DSSA。DSSA 描述在领域模型中表示的需求的解决方案，它不是单个系统的表示，而是能够适应领域中多个系统的需求的一个高层次的设计。建立了领域模型之后，就可以派生出满足这些被建模的领域需求的 DSSA，由于领域模型

中的领域需求具有一定的变化性，DSSA 也要相应地具有变化性。它可以通过表示多选一的（alternative）、可选的（optional）解决方案等来做到这一点。模型和 DSSA 来组织的，因此在这个阶段通过获得 DSSA，也就同时形成了重用基础设施的规约。

3．领域实现

这个阶段的主要目标是依据领域模型和 DSSA 开发和组织可重用信息。这些可重用信息可能是从现有系统中提取得到，也可能需要通过新的开发得到。它们依据领域模型和 DSSA 进行组织，也就是领域模型和 DSSA 定义了这些可重用信息的重用时机，从而支持了系统化的软件重用。这个阶段也可以看作重用基础设施的实现阶段。

值得注意的是，以上过程是一个反复的、逐渐求精的过程。在实施领域工程的每个阶段中，都可能返回到以前的步骤，对以前的步骤得到的结果进行修改和完善，再回到当前步骤，在新的基础上进行本阶段的活动。

5.4.3　参与 DSSA 的人员

参与 DSSA 的人员可以划分为 4 种角色：领域专家、领域分析师、领域设计人员和领域实现人员。

1．领域专家

领域专家可能包括该领域中系统的有经验的用户、从事该领域中系统的需求分析、设计、实现以及项目管理的有经验的软件工程师等。领域专家的主要任务包括提供关于领域中系统的需求规约和实现的知识，帮助组织规范的、一致的领域字典，帮助选择样本系统作为领域工程的依据，复审领域模型、DSSA 等领域工程产品等。

领域专家应该熟悉该领域中系统的软件设计和实现、硬件限制、未来的用户需求及技术走向等。

2．领域分析人员

领域分析人员应由具有知识工程背景的有经验的系统分析员来担任。领域分析人员的主要任务包括控制整个领域分析过程，进行知识获取，将获取的知识组织到领域模型中，根据现有系统、标准规范等验证领域模型的准确性和一致性，维护领域模型。

领域分析人员应熟悉软件重用和领域分析方法；熟悉进行知识获取和知识表示所需的技术、语言和工具；应具有一定的该领域的经验，以便于分析领域中的问题及与领域专家进行交互；应具有较高的进行抽象、关联和类比的能力；应具有较高的与他人交互和合作的能力。

3．领域设计人员

领域设计人员应由有经验的软件设计人员来担任。领域设计人员的主要任务包括控制核个软件设计过程，根据领域模型和现有的系统开发出 DSSA，对 DSSA 的准确性和一致性进行验证，建立领域模型和 DSSA 之间的联系。

领域设计人员应熟悉软件重用和领域设计方法；熟悉软件设计方法；应有一定的该

领域的经验，以便于分析领域中的问题及与领域专家进行交互。

4．领域实现人员

领域实现人员应由有经验的程序设计人员来担任。领域实现人员的主要任务包括根据领域模型和 DSSA，或者从头开发可重用构件，或者利用再工程的技术从现有系统中提取可重用构件，对可重用构件进行验证，建立 DSSA 与可重用构件间的联系。

领域实现人员应熟悉软件重用、领域实现及软件再工程技术；熟悉程序设计；具有一定的该领域的经验。

5.4.4 DSSA 的建立过程

因所在的领域不同，DSSA 的创建和使用过程也各有差异，Tract 曾提出了一个通用的 DSSA 应用过程，这些过程也需要根据所应用到的领域来进行调整。一般情况下，需要用所应用领域的应用开发者习惯使用的工具和方法来建立 DSSA 模型。同时 Tracz 强调了 DSSA 参考体系结构文档工作的重要性。因为新应用的开发和对现有应用的维护都要以此为基础。

DSSA 的建立过程分为 5 个阶段，每个阶段可以进一步划分为一些步骤或子阶段。每个阶段包括一组需要回答的问题，一组需要的输入，一组将产生的输出和验证标准。本过程是并发的（concurrent）、递归的（recursive）、反复的（iterative）。或者可以说，它是螺旋模型（spiral）。完成本过程可能需要对每个阶段经历几遍，每次增加更多的细节。

（1）定义领域范围。本阶段的重点是确定什么在感兴趣的领域中以及本过程到何时结束。这个阶段的一个主要输出是领域中的应用需要满足一系列用户的需求。

（2）定义领域特定的元素：本阶段的目标是编译领域字典和领域术语的同义词词典。在领域工程过程的前一个阶段产生的高层块圈将被增加更多的细节，特别是识别领域中应用间的共同性和差异性。

（3）定义领域特定的设计和实现需求约束：本阶段的目标是描述解空间中有差别的特性。不仅要识别出约束，并且要记录约束对设计和实现决定造成的后果，还要记录对处理这些问题时产生的所有问题的讨论。

（4）定义领域模型和体系结构：本阶段的目标是产生一般的体系结构，并说明构成它们的模块或构件的语法和语义。

（5）产生，搜集可重用的产品单元：本阶段的目标是为 DSSA 增加构件，使它可以被用来产生问题域中的新应用。

DSSA 的建立过程是并发的、递归的和反复进行的。该过程的目的是将用户的需要映射为基于实现限制集合的软件需求，这些需求定义了 DSSA。在此之前的领域工程和领域分析过程并没有对系统的功能性需求和实现限制进行区分，而是统称为“需求”。图 5-15 是 DSSA 的一个三层次系统模型。

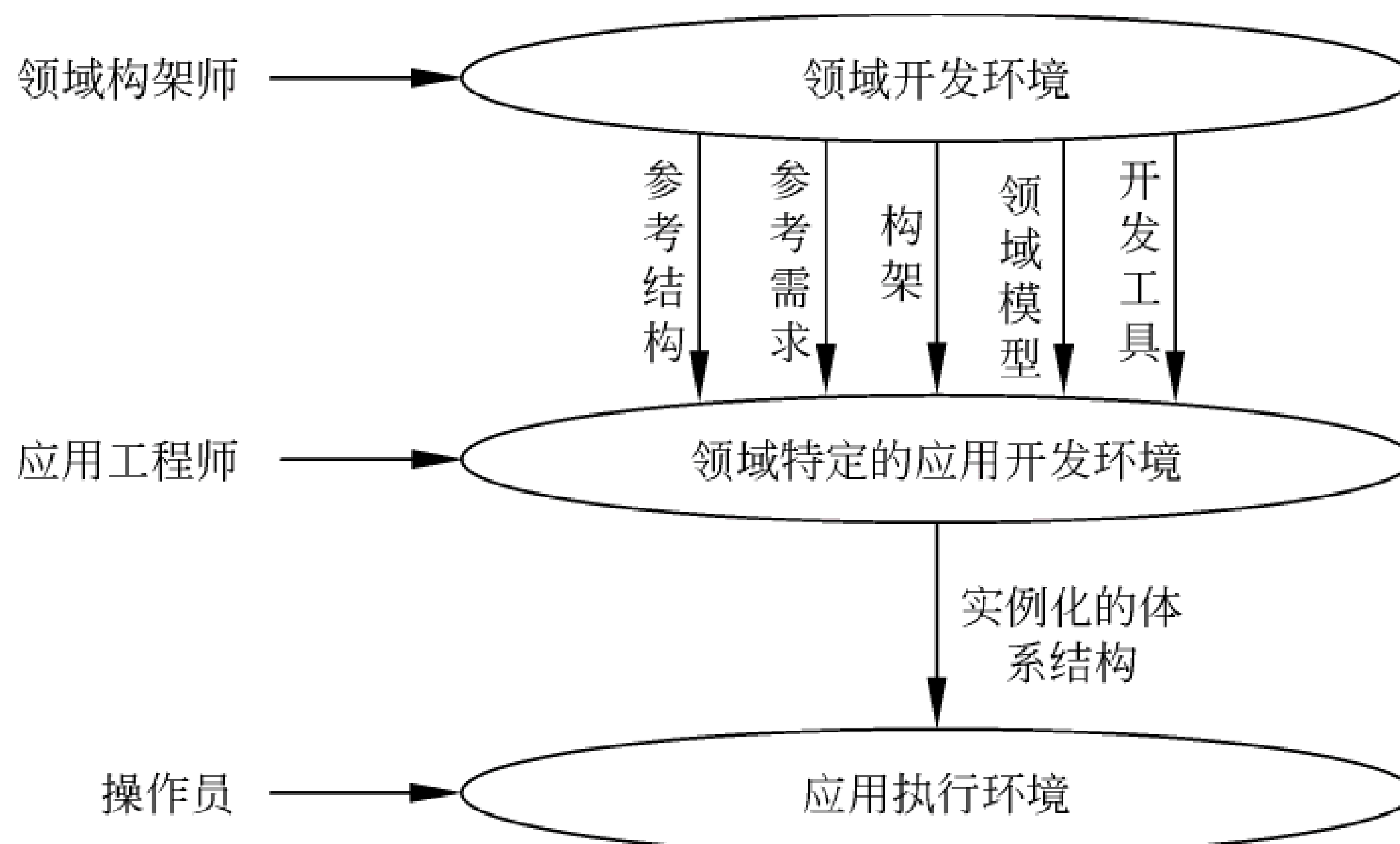


图 5-15　DSSA 的三层次的系统模型

5.5　系统架构的评估

5.5.1　系统架构评估概述

体系结构评估可以只针对一个体系结构，也可以针一对一组体系结构。在体系结构评估过程中，评估人员所关注的是系统的质量属性，所有评估方法所普遍关注的质量属性有以下几个。

1．性能

性能（performance）是指系统的响应能力，即要经过多长时间才能对某个事件做出响应，或者在某段事件内系统所能处理的事件的个数。经常用单位事件内所处理事务的数量或系统完成某个事务处理所需的时间来对性能进行定量的表示。性能测试经常要使用基准测试程序。

2．可靠性

可靠性（reliability）是软件系统在应用或系统错误面前，在意外或错误使用的情况下维持软件系统的功能特性的基本能力。可靠性是最重要的软件特性，通常用它衡量在规定的条件和时间内，软件完成规定功能的能力。可靠性通常用平均失效等待时间（mean time to failure，MTTF）和平均失效间隔时间（mean time between failure，MTBF）来衡量。在失效率为常数和修复时间很短的情况下，MTTF 和 MTBF 几乎相等。可靠性可以分为两个方面。

（1）容错。其目的是在错误发生时确保系统正确的行为，并进行内部“修复”。例如在一个分布式软件系统中失去了一个与远程构件的连接，接下来恢复了连接。在修复这样的错误之后，软件系统可以重新或重复执行进程间的操作直到错误再次发生。

（2）健壮性。这里说的是保护应用程序不受错误使用和错误输入的影响，在遇到意外错误事件时确保应用系统处于已经定义好的状态。值得注意的是，和容错相比，健壮

性并不是说在错误发生时软件可以继续运行，它只能保证软件按照某种已经定义好的方式终止执行。软件体系结构对软件系统的可靠性有巨大的影响。例如，软件体系结构通过在应用程序内部包含冗余，或集成监控构件和异常处理，来支持可靠性。

3．可用性

可用性（availability）是系统能够正常运行的时间比例。经常用两次故障之间的时间长度或在出现故障时系统能够恢复正常的速度来表示。

4．安全性

安全性（security）是指系统在向合法用户提供服务的同时能够阻止非授权用户使用的企图或拒绝服务的能力。安全性是根据系统可能受到的安全威胁的类型来分类的。安全性又可划分为机密性、完整性、不可否认性及可控性等特性。其中，机密性保证信息不泄露给未授权的用户、实体或过程；完整性保证信息的完整和准确，防止信息被非法修改；可控性保证对信息的传播及内容具有控制的能力，防止为非法者所用。

5．可修改性

可修改性（modifiability）是指能够快速地以较高的性能价格比对系统进行变更的能力。通常以某些具体的变更为基准，通过考察这些变更的代价衡量可修改性。可修改性包含以下 4 个方面。

（1）可维护性（maintainability）。这主要体现在问题的修复上：在错误发生后"修复"软件系统。为可维护性做好准备的软件体系结构往往能做局部性的修改并能使对其他构件的负面影响最小化。

（2）可扩展性（extendibility）。这一点关注的是使用新特性来扩展软件系统，以及使用改进版本来替换构件并删除不需要或不必要的特性和构件。为了实现可扩展性，软件系统需要松散耦合的构件。其目标是实现一种体系结构，它能使开发人员在不影响构件客户的情况下替换构件。支持把新构件集成到现有的体系结构中也是必要的。

（3）结构重组（reassemble）。这一点处理的是重新组织软件系统的构件及构件间的关系，例如通过将构件移动到一个不同的子系统而改变它的位置。为了支持结构重组，软件系统需要精心设计构件之间的关系。理想情况下，它们允许开发人员在不影响实现的主体部分的情况下灵活地配置构件。

（4）可移植性（portability）。可移植性使软件系统适用于多种硬件平台、用户界面、操作系统、编程语言或编译器。为了实现可移植，需要按照硬件无关的方式组织软件系统，其他软件系统和环境被提取出。可移植性是系统能够在不同计算环境下运行的能力。这些环境可能是硬件、软件，也可能是两者的结合。在关于某个特定计算环境的所有假设都集中在一个构件中时，系统是可移植的。如果移植到新的系统需要做些更改，则可移植性就是一种特殊的可修改性。

6．功能性

功能性（functionality）是系统所能完成所期望的工作的能力。一项任务的完成需要系统中许多或大多数构件的相互协作。

7．可变性

可变性（changeability）是指体系结构经扩充或变更而成为新体系结构的能力。这种新体系结构应该符合预先定义的规则，在某些具体方面不同于原有的体系结构。当要将某个体系结构作为一系列相关产品（例如，软件产品线）的基础时，可变性是很重要的。

8．互操作性

作为系统组成部分的软件不是独立存在的，经常与其他系统或自身环境相互作用。为了支持互操作性（inter-operation），软件体系结构必须为外部可视的功能特性和数据结构提供精心设计的软件入口。程序和用其他编程语言编写的软件系统的交互作用就是互操作性的问题，这种互操作性也影响应用的软件体系结构。

5.5.2　评估中重要概念

敏感点（sensitivity point）和权衡点（tradeoff point）。敏感点和权衡点是关键的体系结构决策。敏感点是一个或多个构件（和 / 或构件之间的关系）的特性。研究敏感点可使设计人员或分析员明确在搞清楚如何实现质量目标时应注意什么。权衡点是影响多个质量属性的特性，是多个质量属性的敏感点。例如，改变加密级别可能会对安全性和性能产生非常重要的影响。提高加密级别可以提高安全性，但可能要耗费更多的处理时间，影响系统性能。如果某个机密消息的处理有严格的时间延迟要求，则加密级别可能就会成为一个权衡点。

风险承担者（stakeholders）或者称为利益相关人。系统的体系结构涉及很多人的利益，这些人都对体系结构施加各种影响，以保证自己的目标能够实现。表 5-1 列出在体系结构评估中可能涉及的一些风险承担者及其所关心的问题。

表 5-1　系统架构评估中的风险

风险承担者	定义	所关心的问题
系统生产者		
软件系统架构师	负责软件体系结构以及在相互竞争的质量需求间进行权衡的人	对其他风险承担者提出的质量需求的缓解和调停
开发人员	设计人员或程序员	体系结构描述的清晰与完整、各部分的内聚性与受限藕合、清楚的交互机制
维护人员	系统初次部署完成后对系统进行更改的人	可维护性，确定出某个更改发生后必须对系统中哪些地方进行改动的能力
集成人员	负责构件集成和组装的开发人员	与上同
测试人员	负责系统测试的开发人员	集成、一致的错误处理协议，受限的构件耦合、构件的高内聚性、概念完整性
标准专家	负责所开发软件必须满足的标准细节的开发人员	对所关心问题的分离、可修改性和互操作性
性能工程师	分析系统的工作产品以确定系统是否满足其性能及吞吐量需求的人员	易理解性、概念完整性、性能、可靠性

续表

风险承担者	定义	所关心的问题
系统生产者		
安全专家	负责保证系统满足其安全性需求的人员	安全性
项目经理	负责为各小组配置资源、保证开发进度、保证不超出预算的人员，负责与客户沟通	体系结构层次清晰，便于组建小组；任务划分结构、进度标志和最后期限等
产品线经理	设想该体系结构和相关资产怎样在该组织的其他开发中得以重用的人员	可重用性，灵活性
系统消费者		
客户	系统的购买者	开发的进度、总体预算、系统的有用性、满足需求的情况
最终用户	所实现系统的使用者	功能性、可用性
应用开发者（对产品体系结构而言）	利用该体系结构及其他已有可重用构件，通过将其实例化而构建产品的人员	体系结构的清晰性、完整性、简单交互机制、简单裁减机制
任务专家、任务规划者	知道系统将会怎样使用以实现战略目标的客户代表	功能性、可用性、灵活性
系统服务人员		
系统管理员	负责系统运行的人员	容易找到可能出现问题的地方
网络管理员	管理网络的人员	网络性能、可预测性
技术支持人员	为系统在该领域中的使用和维护提供支持的人员	使用性、可服务性、可裁减性
其他人员		
领域代表	类似系统或所考察系统将要在其中运行的系统的构建者或拥有者	可互操作性
系统设计师	整个系统的体系结构设计师，负责在软件和硬件之间进行权衡并选择硬件环境的人	可移植性、灵活性、性能和效率
设备专家	熟悉该软件必须与之交互的硬件的人员，能够预测硬件技术的未来发展趋势的人员	可维护性、性能

场景（scenarios）在进行体系结构评估时，一般首先要精确地得出具体的质量目标，并以之作为判定该体系结构优劣的标准。为得出这些目标而采用的机制叫做场景。场景是从风险承担者的角度对与系统的交互的简短描述。在体系结构评估中，一般采用刺激（stimulus）、环境（environment）和响应（response）三方面来对场景进行描述。

5.5.3　主要评估方法

1．SAAM

SAAM（Scenarios-based Architecture Analysis Method）是卡耐基梅隆大学软件工程研究所（SEI at CMU）的 Kazman 等人于 1983 年提出的一种非功能质量属性的体系结构分析方法，是最早形成文档并得到广泛使用的软件体系结构分析方法。最初它用于比较不同的软件体系的体系结构，以分析 SA 的可修改性，后来实践证明也可用于其他的质量属性如可移植性、可扩充性等，发展成了评估一个系统的体系结构。

（1）特定目标：SAAM 的目标是对描述应用程序属性的文档，验证基本的体系结构假设和原则。此外，该分析方法有利于评估体系结构固有的风险。SAAM 指导对体系结构的检查，使其主要关注潜在的问题点，如需求冲突，或仅从某一参与者的观点出发的不全面的系统设计。SAAM 不仅能够评估体系结构对于特定系统需求的使用能力，也能被用来比较不同的体系结构。

（2）评估技术：SAAM 所使用的评估技术是场景技术。场景代表了描述体系结构属性的基础，描述了各种系统必须支持的活动和将要发生的变化。

（3）质量属性：这一方法的基本特点是把任何形式的质量属性都具体化为场景，但可修改性是 SAAM 分析的主要质量属性。

（4）风险承担者：SAAM 协调不同参与者所感兴趣的方面，作为后续决策的基础，提供了对体系结构的公共理解。

（5）体系结构描述：SAAM 用于体系结构的最后版本，但早于详细设计。体系结构的描述形式应当被所有参与者理解。功能、结构和分配被定义为描述体系结构的三个主要方面。

（6）方法活动：SAAM 的主要输入问题是问题描述、需求声明和体系结构描述。图 5-16 描绘了 SAAM 分析活动的相关输入及评估过程。

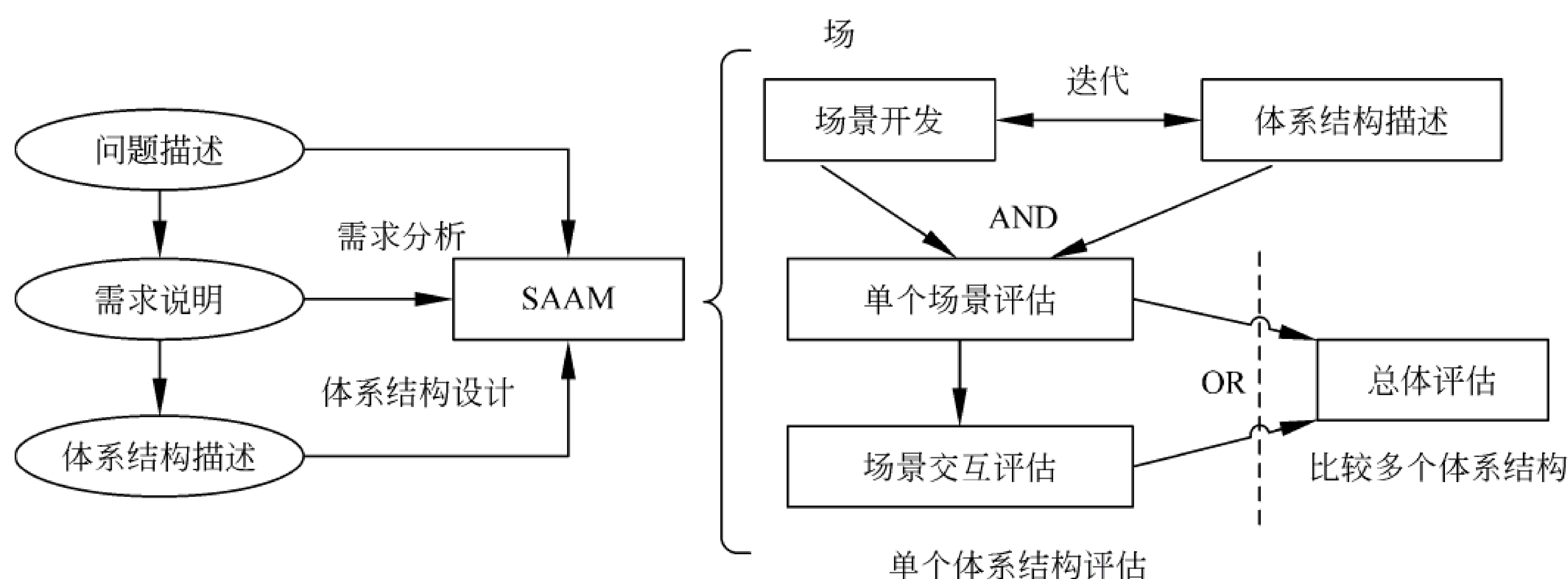


图 5-16　SAAM 输入与评估过程

SAAM 分析评估体系结构的过程包括 5 个步骤，即场景开发、体系结构描述、单个场景评估、场景交互和总体评估。

通过各类风险承担者协商讨论，开发一些任务场景，体现系统所支持的各种活动。

用一种易于理解的、合乎语法规则的体系结构描述 SA，体现系统的计算构件、数据构件以及构件之间的关系（数据和控制）。对场景（直接场景和间接场景）生成一个关于特定体系结构的场景描述列表。通过对场景交互的分析，能得出系统中所有场景对系统中的构件所产生影响的列表。最后，对场景和场景间的交互作一个总体的权衡和评价。

（7）目前知识库的可重用性：SAAM 不考虑这个问题。

（8）方法验证：SAAM 是一种成熟的方法，已被应用到众多系统中，这些系统包括空中交通管制、嵌入式音频系统、WRCS（修正控制系统）、KW IC[8]（根据上下文查找关键词系统）等。

2．ATAM

体系结构权衡分析方法（Architecture Tradeoff Analysis Method，ATAM）是在 SAAM 的基础上发展起来的，主要针对性能、实用性、安全性和可修改性，在系统开发之前，对这些质量属性进行评价和折中。

（1）特定目标：ATAM 的目标是在考虑多个相互影响的质量属性的情况下，从原则上提供一种理解软件体系结构的能力的方法。对于特定的软件体系结构，在系统开发之前，可以使用 ATAM 方法确定在多个质量属性之间折中的必要性。

（2）质量属性：ATAM 方法分析多个相互竞争的质量属性。开始时考虑的是系统的可修改性、安全性、性能和可用性。

（3）风险承担者：在场景、需求收集有关的活动中，ATAM 方法需要所有系统相关人员的参与。

（4）体系结构描述：体系结构空间受到历史遗留系统、互操作性和以前失败的项目约束。在 5 个基本结构的基础上进行体系结构描述，这 5 个结构是从 Kruchten 的 4+1 视图派生而来的。其中逻辑视图被分为功能结构和代码结构。这些结构加上它们之间适当的映射可以完整地描述一个体系结构。

用一组消息顺序图显示运行时的交互和场景，对体系结构描述加以注解。ATAM 方法被用于体系结构设计中，或被另一组分析人员用于检查最终版本的体系结构。

（5）评估技术：可以把 ATAM 方法视为一个框架，该框架依赖于质量属性，可以使用不同的分析技术。它集成了多个优秀的单一理论模型，其中每一个都能够高效、实用地处理属性。该方法使用了场景技术。从不同的体系结构角度，有三种不同类型的场景，分别是用例（包括对系统典型的使用，还用于引出信息）、增长场景（用于涵盖与它的系统修改）、探测场景（用于涵盖那些可能会对系统造成压迫的极端修改）。

ATAM 还使用定性的启发式分析方法（Qualitative Analysis Heuristics），在对一个质量属性构造了一个精确分析模型时要进行分析，定性的启发式分析方法就是这种分析的粗粒度版本。

（6）方法的活动：ATAM 被分为 4 个主要的活动领域（或阶段），分别是场景和需求收集、体系结构视图和场景实现、属性模型构造和分析、折中。图 5-17 描述了与每个阶段相关的步骤，还描述了体系结构设计和分析改进中可能存在的迭代。

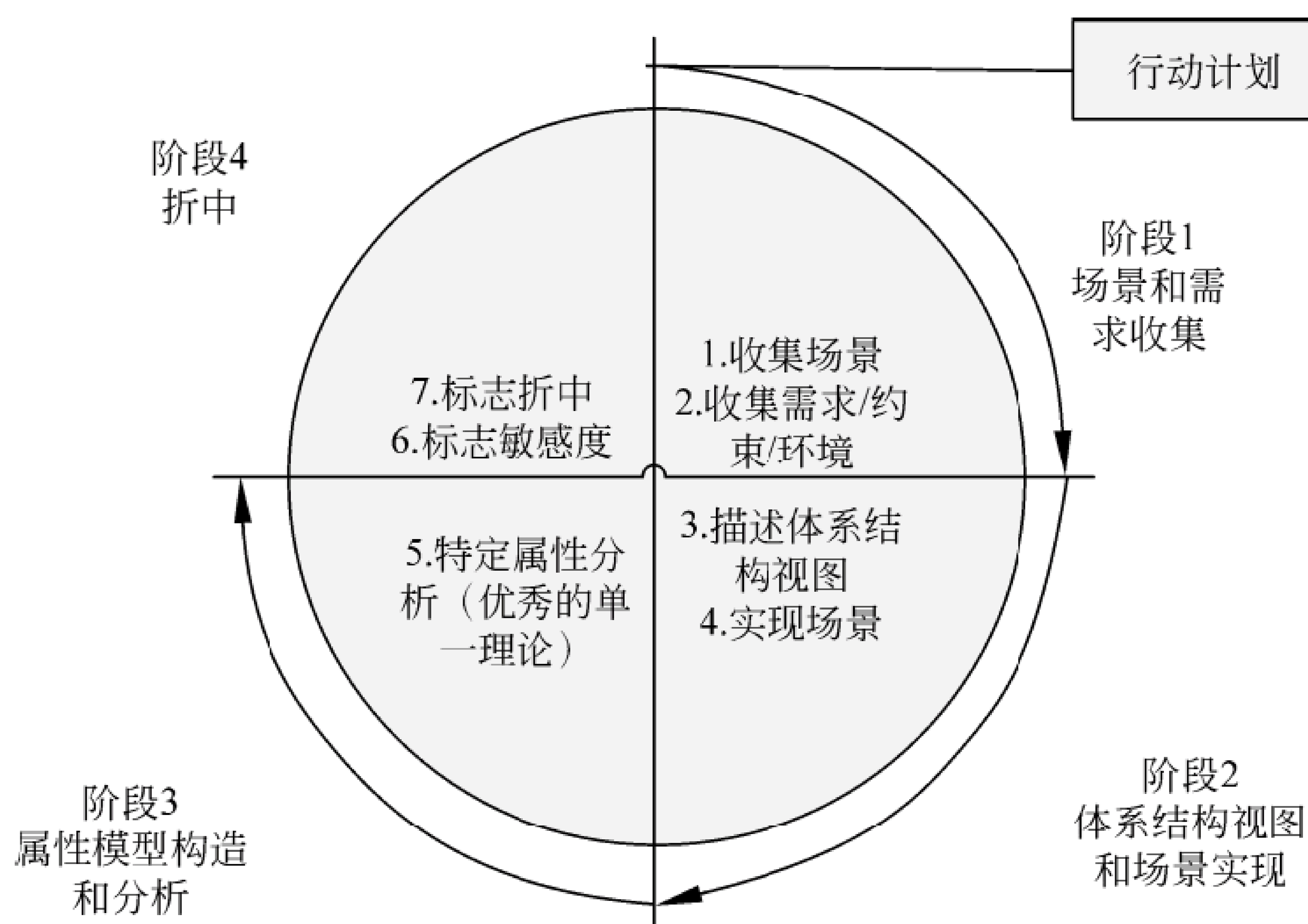


图 5-17　ATAM 分析评估过程

属性专家独立地创建和分析他们的模型，然后交换信息（澄清和创建新的需求）。属性分析是相互依赖的，因为每个属性都会涉及其他属性。获得属性交互的方法有两种，即使用敏感度分析来发现折中点和通过检查假设。

在体系结构设计中，ATAM 提供了迭代的改进。除了通常从场景派生而来的需求，还有很多对行为模式和执行环境的假设。由于属性之间存在着折中，每一个假设都要被检查、验证和提问，以此作为 ATAM 方法的结果。在完成所有这些操作之后，把分析的结果和需求进行比较；如果系统预期的行为大多接近于需求，设计者就可以继续前进，进行下一步更为详细的设计或实现。

（7）领域知识库的可重用性：领域知识库通过基于属性的体系结构风格（Attribute-Based Architectare Style）维护。ABAS 有助于从体系结构风格的概念转向基于特定质量属性模型的推理能力。获取一组基于属性的体系结构风格的目标在于要把体系结构设计变得更为惯例化、更可预测，并得到一个基于属性的体系结构分析的标准问题集合，使设计与分析之间的联系更为紧密。

（8）方法验证：该方法已经应用到多个软件系统，但仍处在研究之中。虽然软件体系结构分析与评价已经取得了很大的进步，但是在某些方面也存在一些问题。例如，体系结构的描述、质量特征的分析、场景不确定性的处理、度量的应用体系结构分析和评价支持工具等，这些都影响和制约着分析评估技术的发展。Clement 认为在未来的 5～10 年内，体系结构的分析是体系结构发展的 5 个方向之一。

第 6 章　UML 建模与架构文档化

早在 20 世纪 70 年代就陆续出现了面向对象的建模方法，在 80 年代末到 90 年代中期，各种建模方法如雨后春笋般从不到 10 种增加到 50 多种。但方法种类的膨胀，同时极大地妨碍了用户的使用和交流。UML（统一建模语言）一出现，以融合了多种面向对象建模方法，简洁的图形与符号，直观的表示和强大的表示能力，得到工业界与学术界认可。它通过统一的表示法，使不同知识背景的领域专家、系统分析和开发人员以及用户可以方便地交流。

6.1　UML 现状与发展

6.1.1　UML 起源

在 1995 年，Gray Booch 和 Janes Rumbaugh 将他们的面向对象建模方法统一为 Unified Method V0.8。一年之后 Ivar Jacobson 加入其中，共同将该方法统一为二义性较少的 UML 0.9。同时，这三位杰出的方法学家被称为“三友（Three Amigos）”。

很快用户也认识到可对软件系统进行可视化、描述、构造和文档化的通用建模语言所带来的益处。他们充满激情地将这种语言的早期草案应用于不同的领域。受用户强烈需求的驱动，建模工具厂商也很快在它们的产品中加入了对 UML 的支持。

UML 成了实际上的工业标准。1996 年，一个由建模专家组成的国际性队伍“UML 伙伴组织”开始同“三友”一起工作，计划将 UML 提议作为 OMG（Object Management Group）的标准建模语言。

1997 年 1 月，伙伴组织向 OMG 提交了最初的提案 UML 1.0。经过了九个月的紧张修订，于 1997 年 9 月提出了最终提案 UML 1.1，这个提案在 1997 年 11 月被 OMG 正式采纳为对象建模标准。

在一个规范被采纳后不久，将成立一个修订任务组，负责该规范的修订。1997 年 9 月，OMG 采纳 UML 1.1 规范之后不久，特许成立了第一个 UML 修订任务组（Revision Task Forces，RTF），负责收集有关评论，并且提出修改建议。

该 RTF 提交的第一个主要产品是一个编辑版本 UML 1.2，它改编了规范，使之与其他 OMG 规范更为一致。尽管这一版本纠正了印刷和语法错误，以及某些明显的逻辑上的不一致，但还是没有涉及对重要技术的改进。

该 RTF 的第二个主要的产品是其技术版本 UML 1.3，它修正和改善了 UML 1.1 的遗留问题，并矫正了在此之后发现的许多小错误。该 RTF 一致推荐 OMG 批准其 UML 1.3

最终草案，并于 1999 年 6 月提交了一份最终报告。被推荐的规范随后被提交给组织委员会和平台技术委员会以获得批准。

6.1.2　UML 体系结构演变

UML 是用元模型来描述的，元模型是 4 层元模型体系结构模式中的一层。此模式的其他层次分别是元-元模型层、模型层和用户对象层。其中元模型层由元-元模型层导出，UML 的元-元模型层在 OMG MOF 的元-元模型中定义，而 UML 元模型中的元类是 MOF 元-元类的实例。

元模型的体系结构模式已被证明可以用来定义复杂模型所要求的精确语义，这种复杂模型通常需要被可靠地保存、共享、操作以及在工具之间进行交换。它的特点如下：

（1）它在每一层都递归地定义语义结构，从而使语义更精确、更正规。

（2）它可用来定义重量级和轻量级扩展机制，如定义新的元类和构造型。

（3）它在体系结构上将 UML 元模型与其他基于 4 层元模型体系结构的标准（比如 MOF 和用于模型交换的 XMI Facility）统一起来。

在元模型层，UML 元模型又被分解为三个逻辑子包：基础包、行为元素包和模型管理包。其中基础包由核心、扩展机制和数据类型三个子包构成，它是描述模型静态结构的语言底层结构，支持类图、对象图和构件图和部署图等结构图。行为元素包是描述模型动态行为的语言上层结构，支持不同的行为图，包括 Use Case（用况）图、顺序图、协作图、状态图和活动图。模型管理包则定义了对模型元素进行分组和管理的语义，它描述了几种分组结构，包括包、模型和子系统。行为元素包和模型管理包都依赖于基础包。

UML 1.3 是建模语言规范第一个成熟的发布。它纠正或调整了从 UML 1.1 中继承下来的遗留问题，并且修正了最终提交后的一年来所发现的大多数错误。从建模者的角度看，从 UML 1.1 到 UML 1.3 并没有很大变化，对语言的大部分改进是在底层对 UML 元模型语义的调整，只有很少量的变化是针对表示法的细枝末节的修改。底层结构上的变化对大多数用户来说是看不到的，但这使得 UML 在将来更容易实现和扩展。

1．解决 UML 1.1 的遗留问题

（1）完善活动图的语义和表示法 增加了状态的动态激发语义，定义了执行条件线程的语义和表示法，而且增加了对象流功能。为了做这些修订，还需要对活动图所依赖的状态机语义做以下修改：为同步并发的活动加入“同步状态”、精化信号的语义、为合并状态转换定义附加的伪状态。

（2）清理关系的标准元素。引入关系元类来组织各种类型的关系，并且把依赖构造型改造为依赖和流。此外，精练了泛化，不再需要以前的许多构造型（如继承、私有、子类、子类型等）。依赖和其他关系名称的一致性也有所改进。

（3）体系结构的一致性。通过加入物理元模型和 XMI（XML metadata Interchange）、

DTD（Document Type Definition）定义，提高了 UML 1.3 元模型的体系结构跟 MOF 和 XMI Facility 的一致性。从 UML 语义逻辑元模型导出的物理元模型包含了一些支持产生 IDL（Interface Definition Language）和 XMI DTD 的修改（例如将关联类转化为类）。尽管这样做与严格的元模型方法相左，但它为未来 UML 的修订达到这一目标提供了桥梁。

2. 其他变化

（1）静态结构图。放宽了限制，使类和接口之间可以关联，并且在类中可以声明信号。信号被定义为一个类元，可以操作。另外，还重新定义了模板和强类型的语义。

（2）用例图。用例之间的关系被重新定义为三种主要类型：泛化、包含和延伸。

（3）交互图。放宽了限制，使用户可以描述角色或实例。而且协作也可以泛化。

（4）模型管理图。改进了模型和子系统的语义和表示法，将它们从包中分离出来，并使之更容易使用。澄清了对包的访问和引入权限的区别。

尽管 UML 规范的核心是语法和语义定义，但它还包括模型交换、语言扩展以及约束等方面的定义。UML 1.3 对这些相关规范都进行了错误纠正，并使之与核心语言的改进保持一致。

3. 为 UML 2.0 确立路标

该 RTF 在最终报告中明确了因为超出其范围或时间不允许而不能做的各种改进。他们建议下一个 RTF 应特别注意扩展性和文档管理方面的问题。对目前的扩展机制，用户和工具开发商已经发现了一些重要问题，而涌入新 UML 外围的提案可能会加剧这些困难。在文档管理方面，物理元模型和 XMI DTD 规范的加入大幅度地增加了 UML 规范的长度，并使它变得笨拙难用（它现在已有 800 多页了）。下一次 UML 修订将会把物理建模规范拆分为单独的文档。

该 RTF 还进一步建议负责起草 UML 2.0 RFP 的工作组考虑以下问题。

- 体系结构：使用严格的元模型方法定义一个与 MOF 元-元模型严格一致的物理元模型。给出改进的指导方针，以决定哪些部分应该定义在核心语言中，哪些部分应定义在 UML 的外围或标准模型库中。
- 扩展性：提供同 4 层元模型体系结构一致的扩展机制。提高外围规范的严密程度，使其支持用户对语言定制能力不断增加的要求。
- 构件：增强基于构件的软件开发的语义和表示法。
- 关系：提供“精化”和“追踪”依赖关系的基本语义。在多个抽象层次上定义关联的语义。
- 状态图和活动图：定义独立于状态图语义的活动图语义。在活动图和状态图中提供更随意的并发。详细说明状态机的泛化。
- 模型管理：重新定义模型和子系统的表示法和语义，以增强对企业体系结构视图的支持。
- 总体机制：定义一种模型版本管理的机制。详细说明图的互换机制。

6.1.3　UML 的应用与未来

UML 是在多种面向对象建模方法的基础上发展起来的建模语言，主要用于软件密集型系统的建模。它的演化，可以按其性质划分为以下几个阶段：最初的阶段是专家的联合行动，由三位 Object-Oriented（面向对象）方法学家将他们各自的方法结合在一起，形成 UML 0.9。第二阶段是公司的联合行动，由十几家公司组成的“UML 伙伴组织”将各自的意见加入 UML，形成 UML 1.0 和 1.1，并作为向 OMG 申请成为建模语言规范的提案。第三阶段是在 OMG 控制下的修订与改进，OMG 于 1997 年 11 月正式采纳 UML 1.1 作为建模语言规范，然后成立任务组进行不断的修订，并产生了 UML 1.2、1.3 和 1.4 版本，其中 UML 1.3 是较为重要的修订版。目前正处于 UML 的重大修订阶段，目标是推出 UML 2.0，作为向 ISO 提交的标准提案。

从 UML 的早期版本开始，便受到了计算机产业界的重视，OMG 的采纳和大公司的支持把它推上了实际上的工业标准的地位，使它拥有越来越多的用户。它被广泛地用于应用领域和多种类型的系统建模，如管理信息系统、通信与控制系统、嵌入式实时系统、分布式系统和系统软件等。近几年还被运用于软件再工程、质量管理、过程管理和配置管理等方面。而且它的应用不仅仅限于计算机软件，还可用于非软件系统，例如硬件设计、业务处理流程、企业或事业单位的结构与行为建模。

对 UML 的讨论和评价，无论是 Internet 上的意见交流，或是每年一次的 UML 研讨会，还是学术期刊上发表的文章，都是既肯定其成绩，又指出其缺点和错误，并且以积极的态度提出建设性意见。总的来说：

- UML 已经取得重要成功，它已成为在软件工业中占支配地位的建模语言，并在许多领域的软件开发中得到应用。
- UML 还存在许多问题，自它产生之日起就从未离开过批评：用户和教师抱怨它内容庞大、难学难教而且太过复杂；学者认为它缺少一个精练的核心和定义良好的外围，有些语义定义得不够精确而且带有二义性；建模实践者认为它缺少支持自己领域建模要求的机制；工具开发商则因为规范本身的不确定性而产生理解上的偏差，它们对 UML 的自行诠释有可能误导用户。
- UML 的关键问题是过于庞大和复杂，以及在语言体系结构、语义等方面存在理论缺陷。产生这些问题的一个重要原因是，在形成规范的过程中不得不照顾多种方法流派的观点和多家公司的利益。

6.2　UML 基础

6.2.1　概述

UML 通过图形化的表示机制从多个侧面对系统的分析和设计模型进行刻画。它共

定义了 10 种视图，并将其分为如下 4 类。

（1）用例图（use case diagram）。从外部用户的角度描述系统的功能，并指出功能的执行者。

（2）静态图。包括类图（class diagram）、对象图（object diagram）和包图（package diagram）。类图描述系统的静态结构，类图的节点表示系统中的类及其属性和操作，类图的边表示类之间的联系，包括继承、关联、依赖和聚合等。对象图是类图的一个实例，它描述在某种状态下或在某一时间段，系统中活跃的对象及其关系。包图描述系统的分解结构，它表示包（package）以及包之间的关系。包由子包及类组成。包之间的关系包括继承、构成与依赖关系。

（3）行为图。包括交互图（interactive diagram）、状态图（statechart diagram）与活动图（active diagram），它们从不同的侧面刻画系统的动态行为。交互图描述对象之间的消息传递，它又可分为顺序图（sequence diagram）与合作图（collaboration diagram）两种形式。顺序图强调对象之间消息发送的时间序。合作图更强调对象间的动态协作关系。合作图也可通过消息序号来表示消息传递的时间序，只不过这种表示不如顺序图那样直观。状态图描述类的对象的动态行为，它包含对象所有可能的状态、在每个状态下能够响应的事件以及事件发生时的状态迁移与响应动作。活动图描述系统为完成某项功能而执行的操作序列，这些操作序列可以并发和同步。活动图中包含控制流和信息流。

（4）实现图（implementation diagram）。包括构件图（component diagram）与部署图（deployment diagram），它们描述软件实现系统的组成和分布状况。构件图描述软件实现系统中各组成部件以及它们之间的依赖关系。部署图描述作为软件系统运行环境的硬件及网络的物理体系结构，其节点表示实际的计算机和设备，边表示节点之间的物理连接关系，也可显示连接的类型及节点之间的依赖性。

6.2.2 用例和用例图

用例（use case）国内也翻译为用况、用案等，在 UML 中，用例用一个椭圆表示，用例名往往用动宾结构或主谓结构命名。它有两个比较有代表性的定义如下。

定义 1：用例是对一个活动者（actor）使用系统的一项功能时所进行的交互过程的一个文字描述序列。

定义 2：用例是系统、子系统或类和外部的参与者（actor）交互的动作序列的说明，包括可选的动作序列和会出现异常的动作序列。

用例是代表系统中各相关人员之间就系统的行为所达成的契约。软件的开发过程可以分为需求分析、设计、实现等阶段，在需求阶段用例是分析人员与客户沟通的工具和项目规模估算的依据；设计阶段用例是系统功能设计的主要输入；在实现阶段用例是检测类行为正确性的文档。因此，面向对象的软件开发过程是用例驱动的。

用例分析可以支持领域建模（domain modeling），以确保定义正确的需求（right

requirements），是保证 OO 软件开发成功的基础。但要在具体的项目中灵活使用用例来捕获用户的需求并不是一件容易的事情，往往需要用户的经验、沟通能力、丰富的领域知识等。

本质上，用例分析是一种功能分解（functional decomposition）的技术，并未使用到面向对象思想。但用例是 UML 的重要部分，确定一个系统的用例是开发 OO 系统的第一步，用例分析这步做得好，接着的交互图分析、类图分析等才有可能做得好，整个系统的开发才能顺利进行。

编写用例必须识别以下元素。

1）参与者

角色（actor）是指系统以外的、需要使用系统或与系统交互的东西，包括人、设备、外部系统等。actor 有很多不同的译名，包括参与者、活动者、执行者和行动者等。

一个参与者可以执行多个用例，一个用例也可以由多个参与者使用。但需要注意的是，参与者实际上并不是系统的一部分，尽管在模型中会使用参与者。

参与者实际上是一个版型化的类，其版型是<<Actor>>。图 6-1 是参与者的三种表示形式。

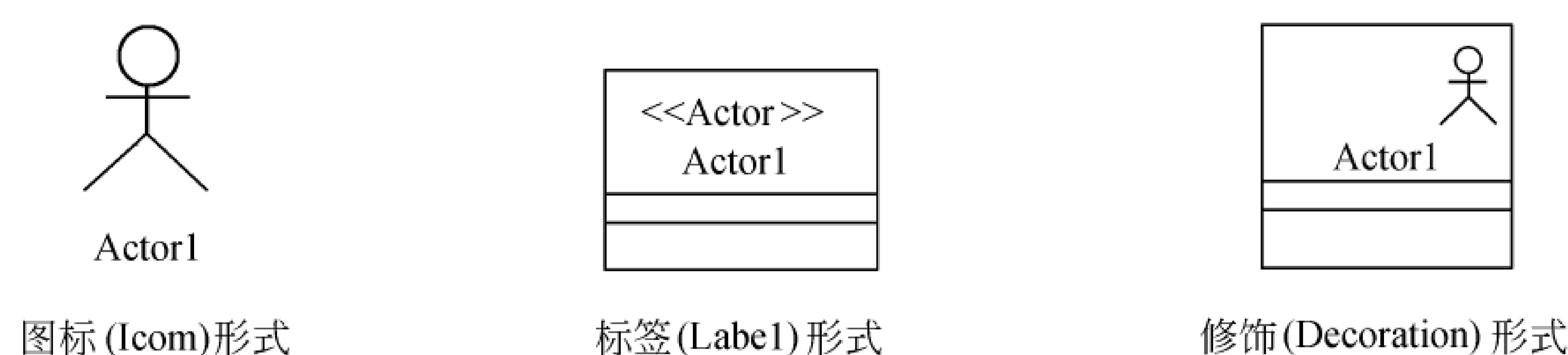


图 6-1　参与者的表示形式

2）用例间的关系

用例除了与参与者有关联（association）关系外，用例之间也存在着一定的关系（relationship），如泛化（generalization）关系、包含（include）关系、扩展（extend）关系等。

包含（include）关系指的是两个用例之间的关系，其中一个用例（称作基本用例，base use case）的行为包含了另一个用例（称作包含用例，inclusion use case）的行为。包含关系是依赖关系的版型，也就是说包含关系是比较特殊的依赖关系，它们比一般的依赖关系多一些语义。如图 6-2 所示是包含关系的例子，其中用

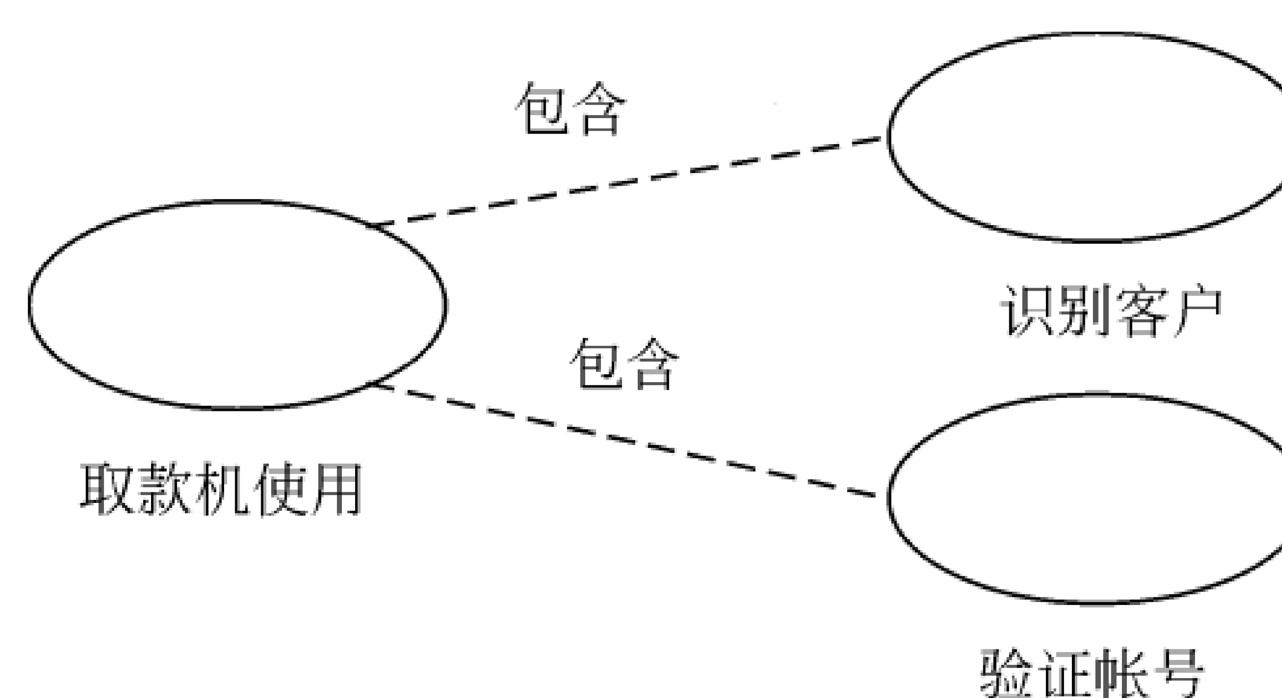


图 6-2　用例的包

例取款机专用（ATM Session）是基本用例，用例识别客户（Identify Customer）和验证账号（Validate Account）是包含用例。

扩展（extend）关系的基本含义与泛化关系类似。但在扩展关系中，对于扩展用例（extension use case）有更多的规则限制，即基本用例必须声明若干“扩展点”（extension point），而扩展用例只能在这些扩展点上增加新的行为和含义。图 6-3 所示是同时具有扩展关系和包含关系的例子，在这个例子中，可以看到基本用例、包含用例、扩展用例等概念间的联系和区别。

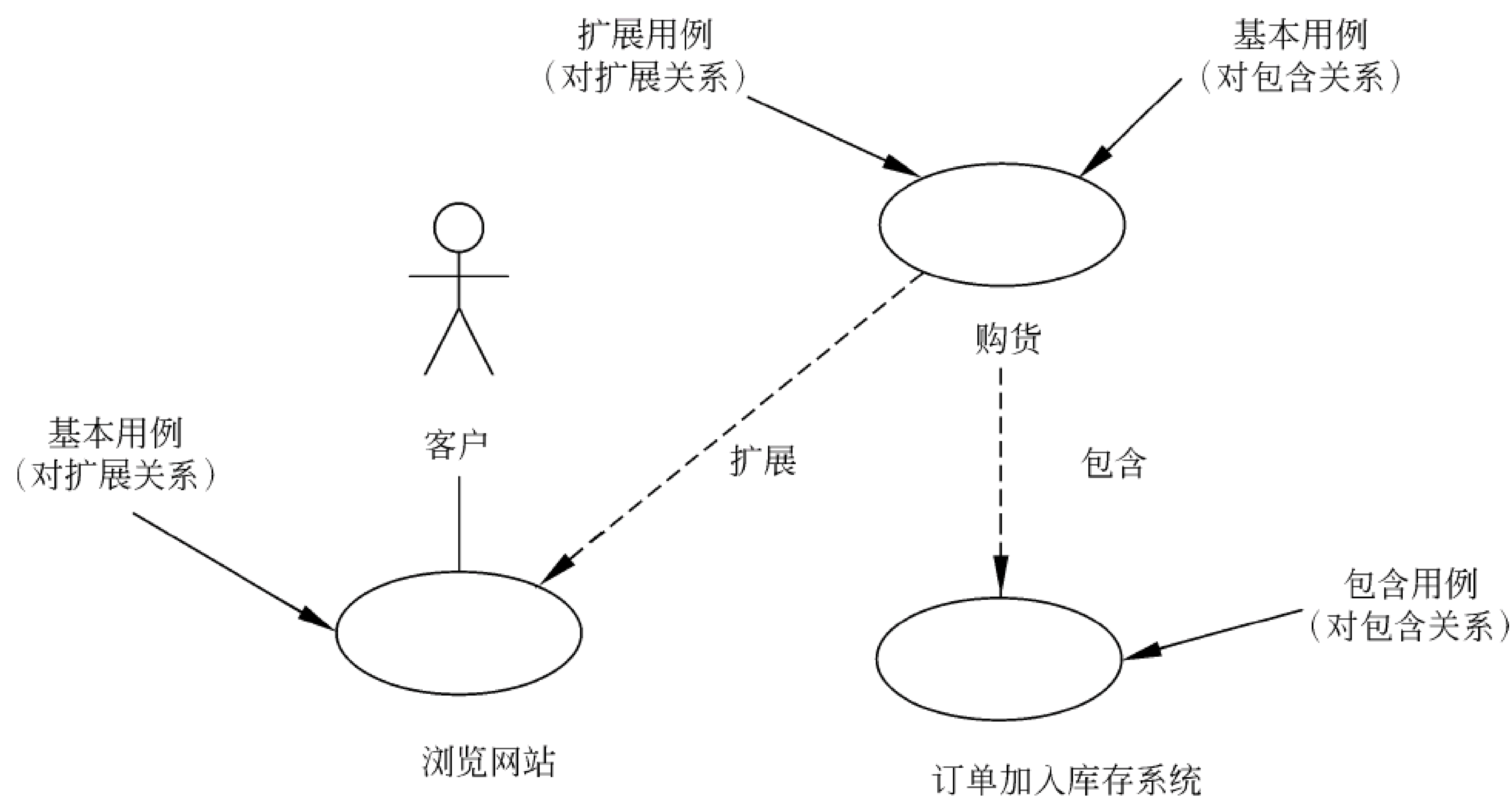


图 6-3　包含用例和扩展用例

对于“购货”这个用例，它扩展了“浏览网站”这个用例，同时也包含了“订单加入库存系统”这个用例。因此对于“浏览网站”这个用例来说是扩展用例，但对于“订单加入库存系统”这个用例来说是基本用例。

3）用例图

用例图（use case diagram）是显示一组用例、参与者以及它们之间关系的图。在 UML 中，一个用例模型由若干个用例图描述。如图 6-4 显示了电话系统的使用用例图。

UML 规范说明中并不使用颜色作为图形语义的区分标记，但建模人员可以在图中给某些图符加上填充颜色，以强调某一部分的模型，或希望引起使用者的特别注意。但在语义上，使用填充颜色和不使用填充颜色的模型是一样的。

4）用例的描述

用例的描述才是用例的核心部分，用例采用自然语言描述参与者与系统进行交互时双方的行为，不追求形式化的语言表达。以下是一个典型描述多方式。

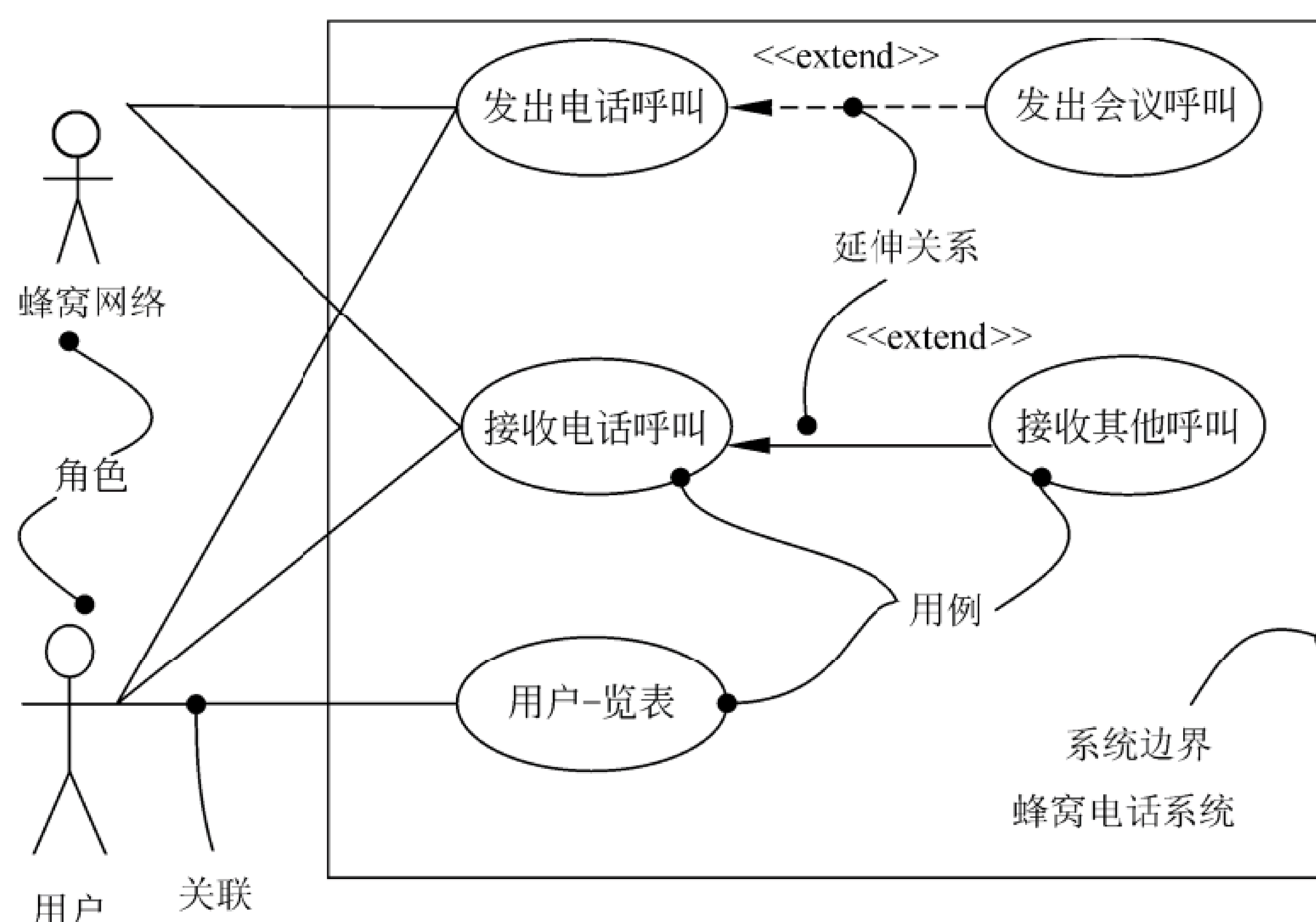


图 6-4　电话系统的使用用例图

用例：<编号><名称>
用途及特征：
用例在系统中的目标（用例目标描述）
范围（当前考虑的是哪个系统）
级别（概要任务/首要任务/子功能）
当前条件（用例执行前系统应具有的状态）
成功后续条件（用例成功执行后应具有的状态）
失效后续条件（用例没有完成目标的状态）
触发（启动该用例执行的系列动作）
角色：首要角色（与该用例关联的首要角色）
主场景：动作序列
<步骤编号><动作描述><系统响应>
扩展场景：动作序列
<步骤编号><条件>:<动作或另一个用例>
异常场景：
<步骤编号><条件>:<异常动作>
相关信息（可选）：
优先级（该用例对于系统/组织的关键程度）
性能目标（该用例的执行时间耗费）
频度（该用例被执行的频度）
与首要角色的联系渠道（包括交互式、静态文件、数据库等）
存在问题：
列出关于该用例的未解决问题

6.2.3 交互图

交互图（interaction diagram）是用来描述对象之间以及对象与参与者（actor）之间的动态协作关系以及协作过程中行为次序的图形文档。它通常用来描述一个用例的行为，显示该用例中所涉及的对象和这些对象之间的消息传递。交互图包括顺序图（sequence diagram）和协作图（collaboration diagram）两种形式。顺序图着重描述对象按照时间顺序的消息交换，协作图着重描述系统成分如何协同工作。顺序图和协作图从不同的角度表达了系统中的交互和系统的行为，它们之间可以相互转化。一个用例需要多个顺序图或协作图，除非特别简单的用例。

交互图可以帮助分析人员对照检查每个用例中所描述的用户需求，如这些需求是否已经落实到能够完成这些功能的类中去实现，提醒分析人员去补充遗漏的类或方法。

1．顺序图

顺序图也称时序图。Rumbaugh 对顺序图的定义是：顺序图是显示对象之间交互的图，这些对象是按时间顺序排列的。特别地，顺序图中显示的是参与交互的对象及对象之间消息交互的顺序。图 6-5 所示是一个简单的顺序图例子。

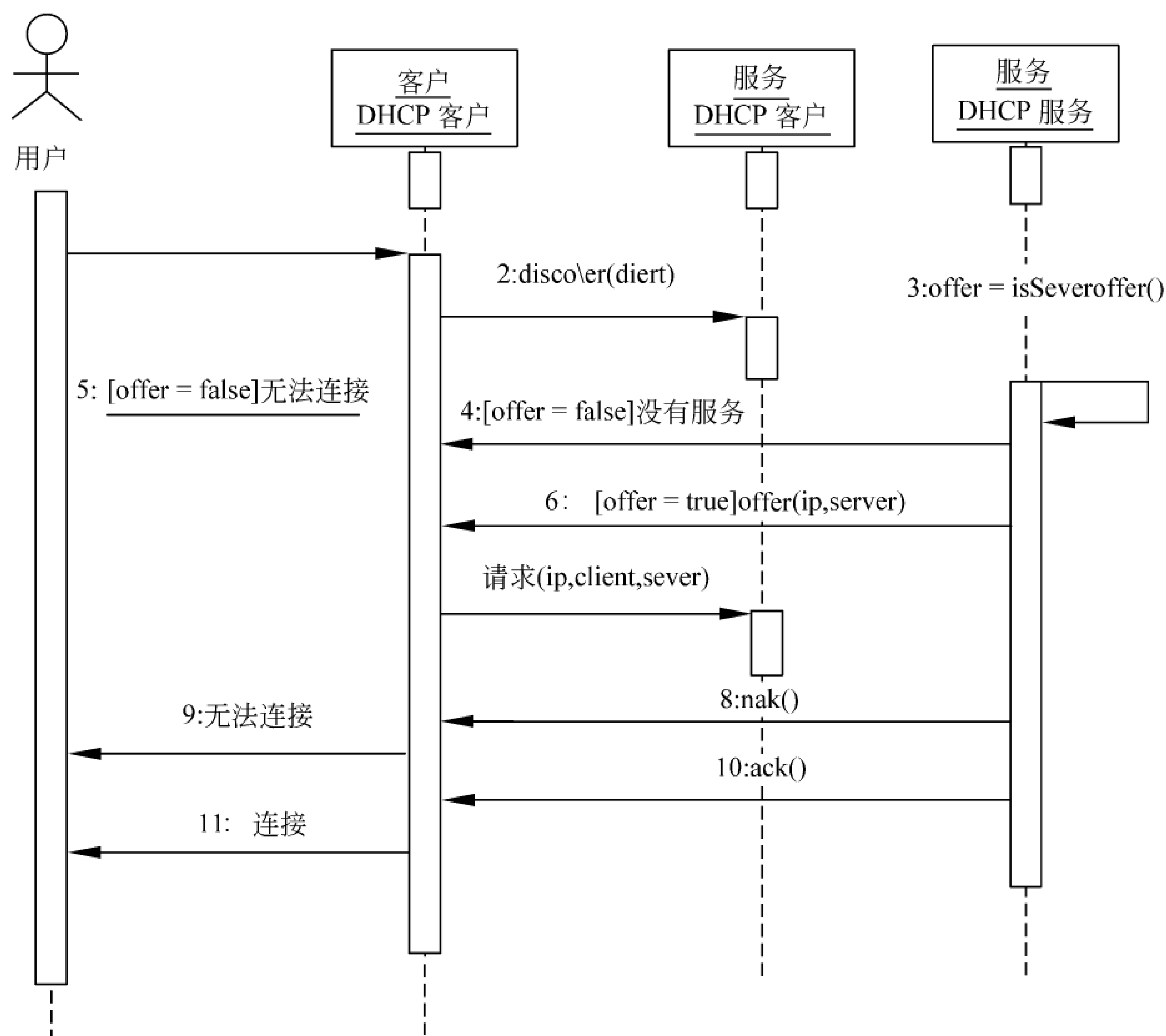


图 6-5 顺序图

顺序图是一个二维图形。在顺序图中水平方向为对象维，沿水平方向排列的是参与交互的对象。其中对象间的排列顺序并不重要，但一般把表示参与者的对象放在图的两侧，主要参与者放在最左边，次要参与者放在最右边（或表示人的参与者放在最左边，表示系统的参与者放在最右边）。顺序图中的垂直方向为时间维，沿垂直向下方向按时间递增顺序列出各对象所发出和接收的消息。

2．协作图

协作图是用于描述系统的行为是如何由系统的成分协作实现的图，协作图中包括的建模元素有对象（包括参与者实例、多对象、主动对象等）、消息、链等。

6.2.4　类图和对象图

类是具有相似结构、行为和关系的一组对象的抽象。在 UML 中，类表示为划分成三个格子的长方形，如图 6-6 所示。

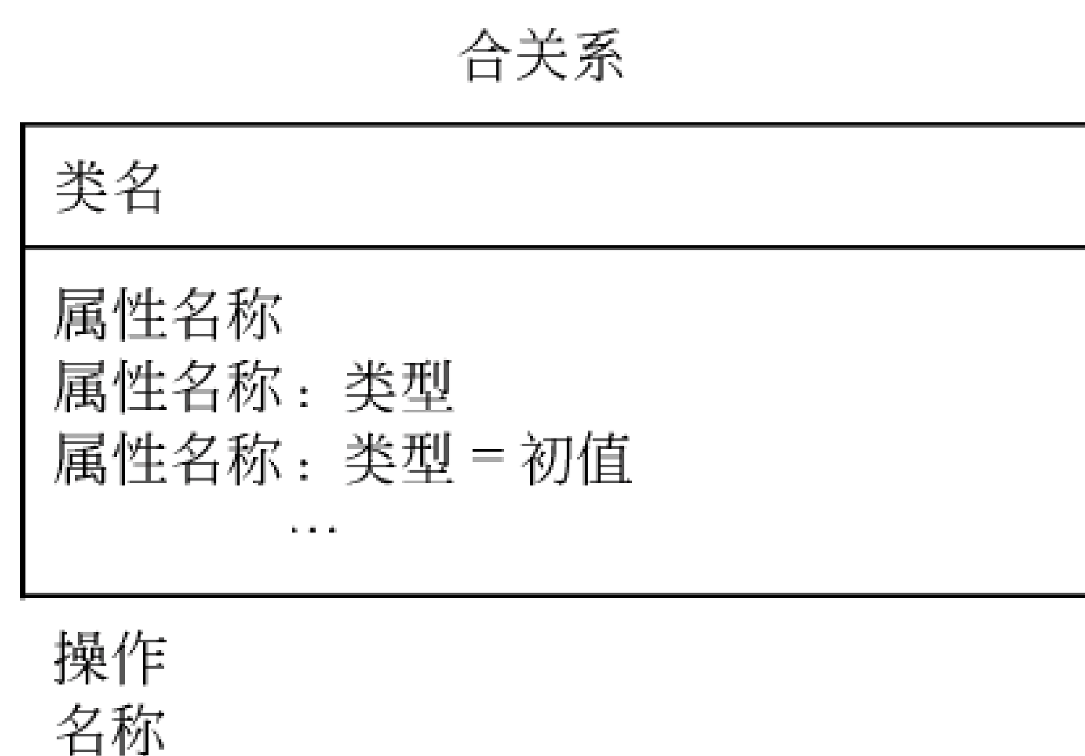


图 6-6　UML 中类的表示图

在定义类的时候，类的命名应尽量用应用领域中的术语，应明确、无歧义，以利于开发人员与用户之间的相互理解和交流。一般而言，类的名字是名词。

一般说来，类之间的关系有关联、聚集、组合、泛化和依赖等，下面将对这些关系进行详细说明。

1）关联

关联（association）是模型元素间的一种语义联系，它是对具有共同的结构特性、行为特性、关系和语义的链（link）的描述。在上面的定义中，需要注意链这个概念，链是一个实例，就像对象是类的实例一样，链是关联的实例，关联表示的是类与类之间的关系，而链表示的是对象与对象之间的关系。

在类图中，关联用一条把类连接在一起的实线表示。关联两端的类可以某种角色参与关联。例如在图 6-7 中，Company 类以 employer 的角色、Person 类以 employee 的角色参与关联，employer 和 employee 称为角色名。如果在关联上没有标出角色名，则隐含地用类的名称作为角色名。角色还具有多重性（multiplicity），表示可以有多少个对象参与该关联。在图 6-7 中，employer 可以雇佣多个 employee，表示为 0..n；employee 只能被一家 employer 雇佣，表示为 1。



图 6-7　关联的角色

通过关联类（association class）可以进一步描述关联的属性、操作以及其他信息。关联类通过一条虚线与关联连接。图 6-8 中的 Contract 类是一个关联类，Contract 类中有属性 salary，这个属性描述的是 Company 类和 Person 类之间的关联的属性，而不是描述

Company 类或 Person 类的属性。

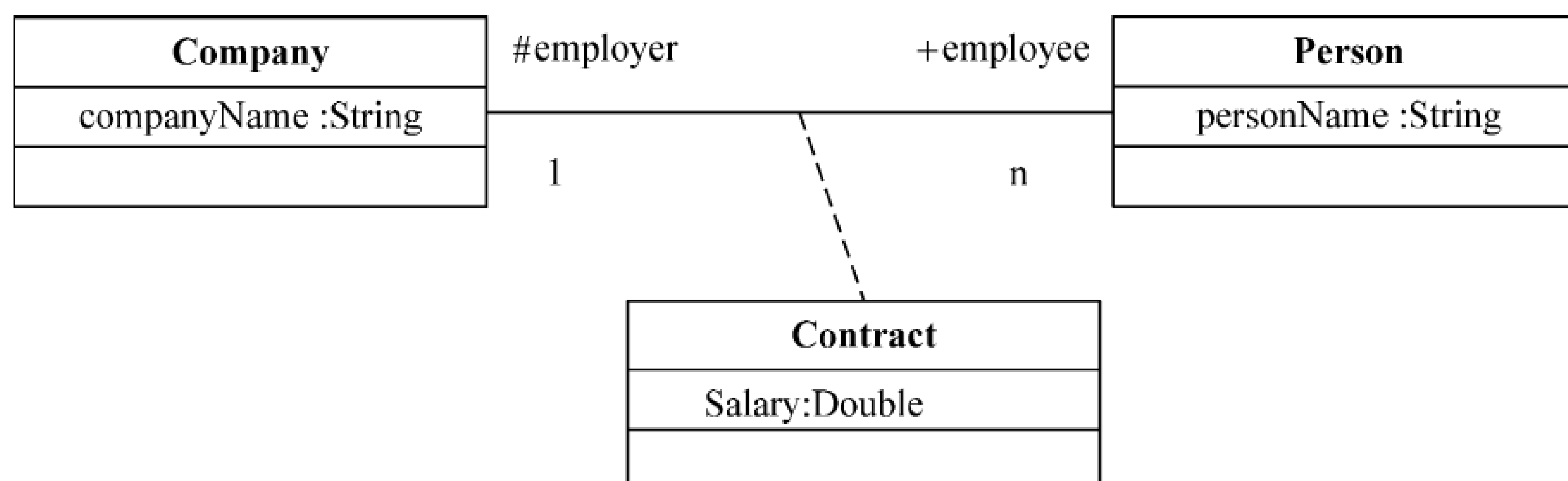


图 6-8 使用关联类的关联

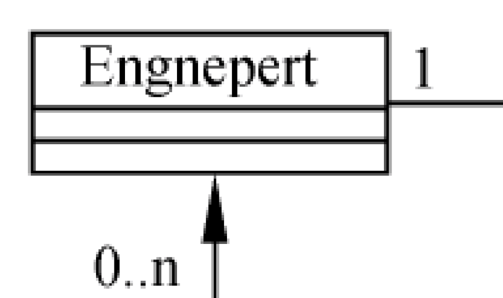


图 6-9 自返关联

自返关联（reflexive association）又称递归关联（recursive association），是一个类与自身的关联，即同一个类的两个对象间的关系。自返关联虽然只有一个被关联的类，但有两个关联端，每个关联端的角色不同。自返关联的例子如图 6-9 所示。

2）聚集和组合

聚集（aggregation）是一种特殊形式的关联。聚集表示类之间整体与部分的关系。在对系统进行分析和设计时，需求描述中的“包含”、“组成”、“分为……部分”等词常常意味着存在聚集关系（见图 6-10）。

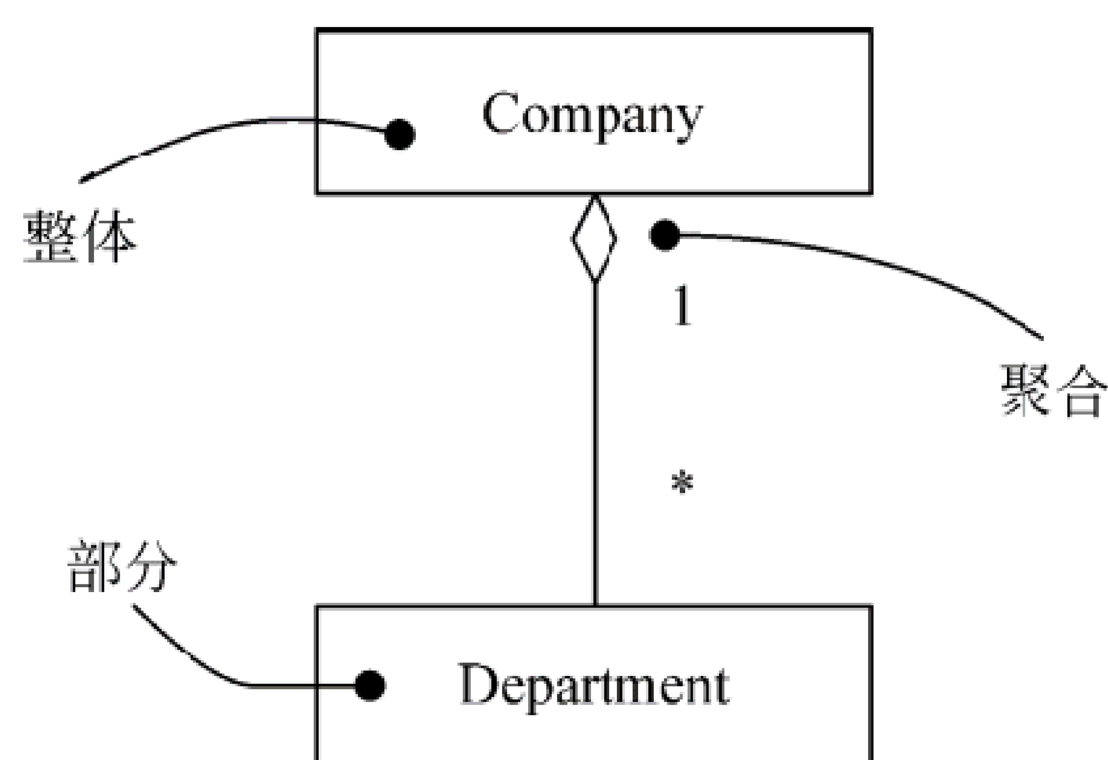


图 6-10 聚集关联

组合（composition）表示的也是类之间的整体与部分的关系，但组合关系中的整体与部分具有同样的生存期。也就是说，组合是一种特殊形式的聚集。

3）泛化关系

泛化（generalization）定义了一般和特殊元素之间关系，如果从面向对象程序设计语言的角度来说，类与类之间的泛化关系就是平常所说的类与类之间的继承关系。

UML 中用一头为空心三角形的连线表示泛化关系。

4）依赖关系

假设有两个元素 X、Y，如果修改元素 X 的定义可能会导致对另一个元素 Y 的定义的修改，则称元素 Y 依赖于元素 X。

5）类图

图 6-11 所示为学校内主要对象的类图。学校包含若干学生，是由多个系组成。每个系开设若干课程，学生参加不同课程学习（管联）关系；老师教一门或多门课程。在一个系中，有一个老师是领导，系包含若干老师。

类图以直观、抽象形式展示了不同对象之间关系。

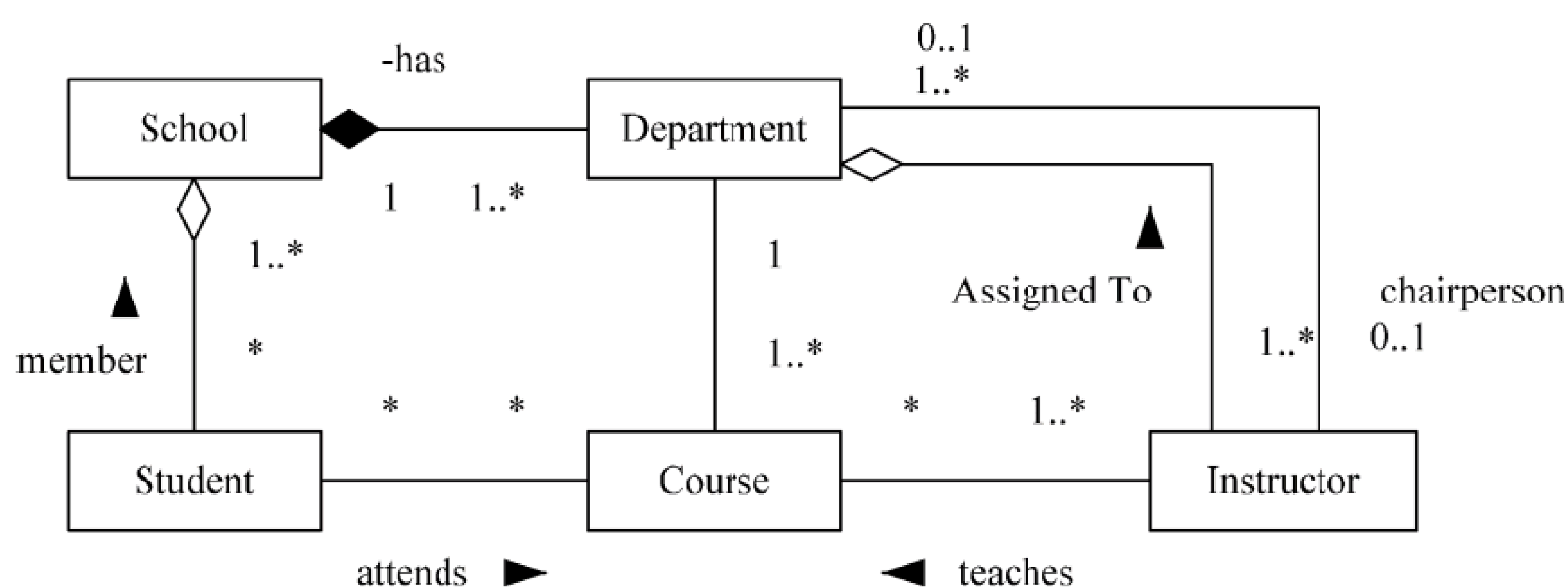


图 6-11　类到需求

6.2.5　状态图和活动图

1．状态图

UML 中的状态图（state chart diagram）主要用于描述一个对象在其生存期间的动态行为，表现一个对象所经历的状态序列，引起状态转移的事件（event），以及因状态转移而伴随的动作（action）。状态图是 UML 中对系统的动态行为建模的 5 个图之一，状态图在检查、调试和描述类的动态行为时非常有用。一般可以用状态机对一个对象的生命周期建模，状态图是用于显示状态机的，重点在于描述状态之间的控制流。

图 6-12 所示是一个简单的状态图的例子。这个状态图中描述的对象除了初态和终态外，还有 Idle 和 Running 两个状态，而 keyPress、finished、shut Down 等是事件。

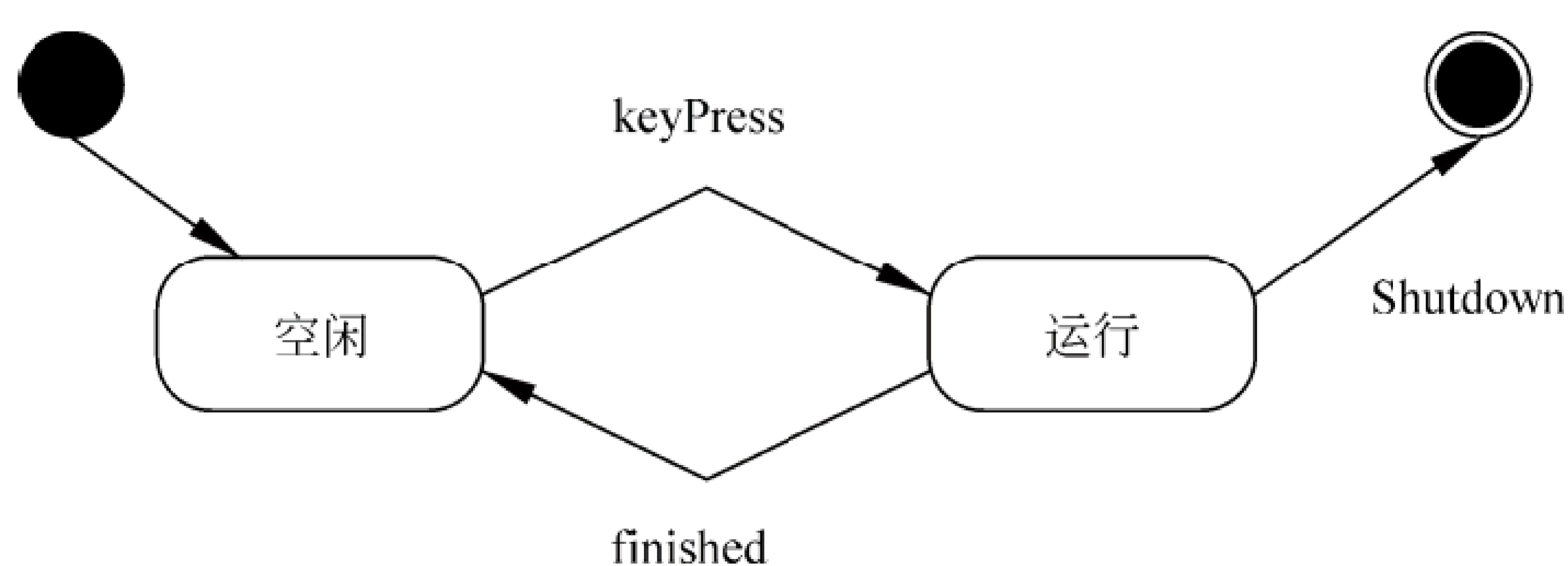


图 6-12　状态图的例子

2．活动图

活动图是对系统的动态行为建模的 5 个图之一。活动图可以用于描述系统的工作流程和并发行为。活动图其实可看作状态图的特殊形式，活动图中一个活动结束后将立即进入下一个活动（在状态图中状态的转移可能需要事件的触发）。

下面讨论活动图中的几个基本概念：活动、泳道、分支、分叉和汇合、对象流。

1）活动

活动（activity）表示的是某流程中的任务的执行，它可以表示某算法过程中语句的执行。在活动图中需要注意区分动作状态（action state）和活动状态（activity state）这两个概念。动作状态是原子的，不能被分解，没有内部转移，没有内部活动，动作状态的工作所占用的时间是可忽略的。动作状态的目的是执行进入动作（entry action），然后转向另一个状态。活动状态是可分解的，不是原子的，其工作的完成需要一定的时间。可以把动作状态看作活动状态的特例。

2）泳道

泳道（swimlane）是活动图中的区域划分，根据每个活动的职责对所有活动进行划分，每个泳道代表一个责任区。泳道和类并不是一一对应的关系，泳道关心的是其所代表的职责，一个泳道可能由一个类实现，也可能由多个类实现。

3）分支

在活动图中，对于同一个触发事件，可以根据不同的警戒条件转向不同的活动，每个可能的转移是一个分支（branch）。

4）分叉和汇合

分支表示的是从多种可能的活动转移中选择一个，如果要表示系统或对象中的并发行为，则可以使用分叉（fork）和汇合（join）这两种建模元素。分叉表示两个或多个控制流经过分叉后，这些控制流并发进行；汇合正好与分叉相反。

5）对象流

在活动图中可以出现对象。对象可以作为活动的输入或输出。活动图中的对象流表示活动和对象之间的关系，如一个活动创建对象（作为活动的输出）或使用对象（作为活动的输入）等。如图 6-13 所示。

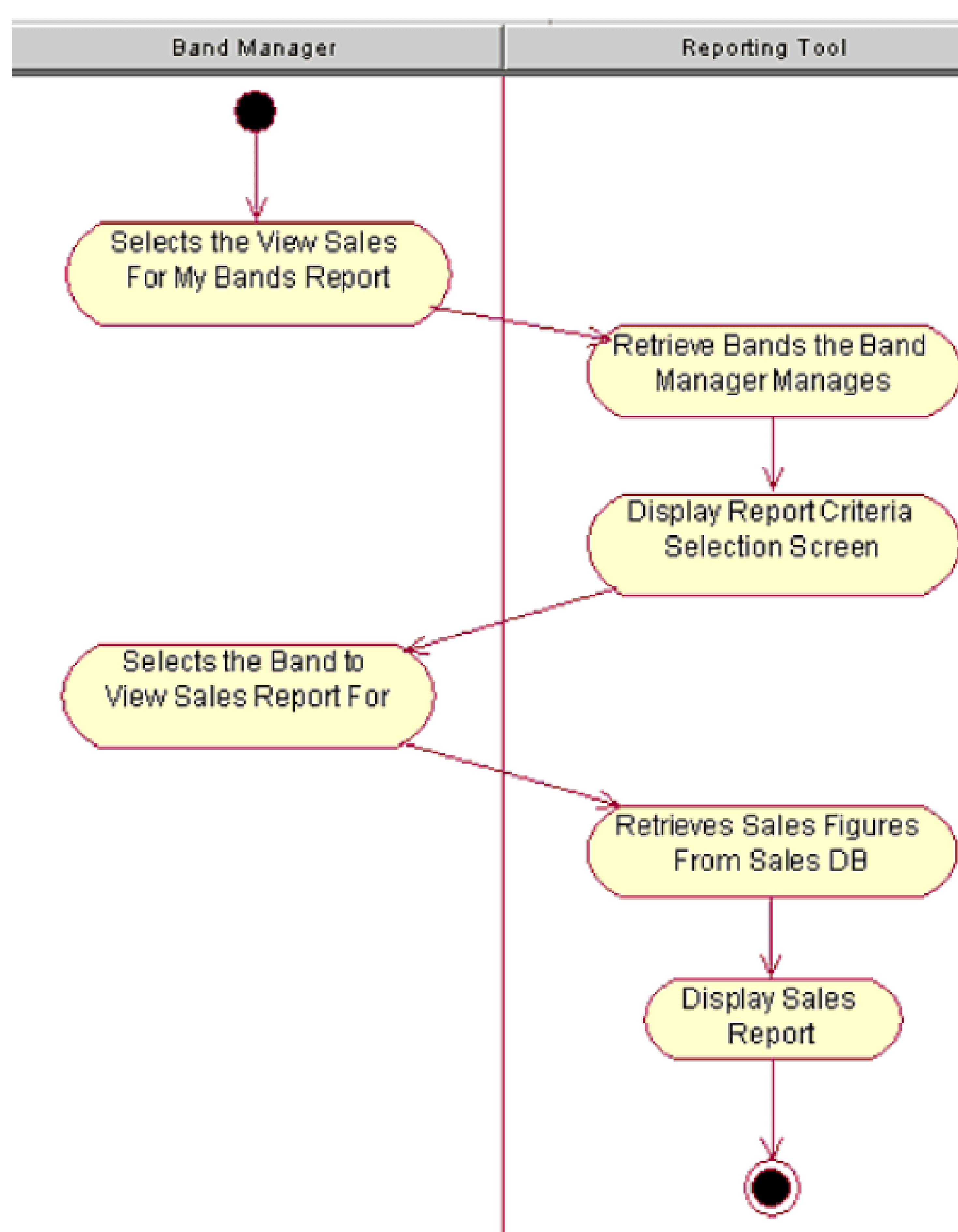


图 6-13　活动图案例

6.2.6　构件图

构件（component）是系统中遵从一组接口且提供其实现的物理的、可替换的部分。

构件图（componentdiagram）则显示一组构件以及它们之间的相互关系，包括编译、链接或执行时构件之间的依赖关系。图 6-14 所示是一个构件图的例子，表示.html 文件、.exe 文件、.dll 文件这些构件之间的相互依赖关系。

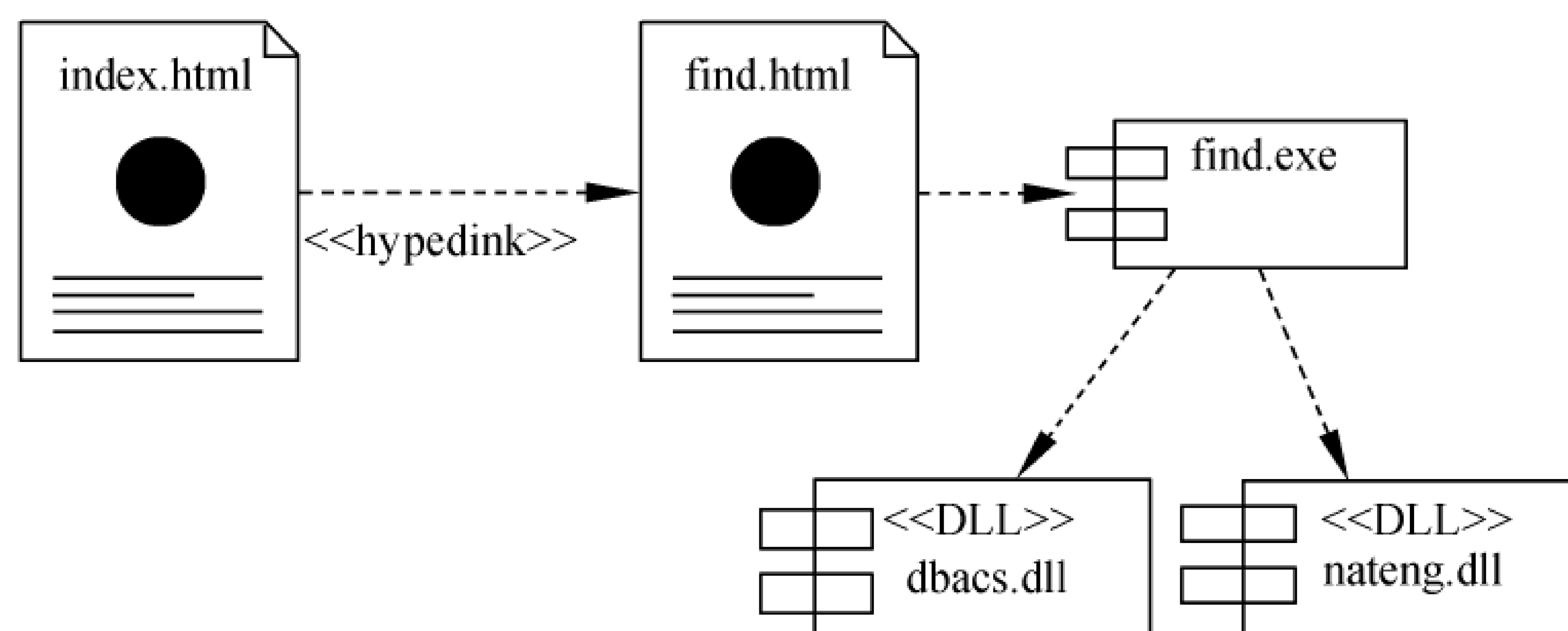


图 6-14　构件图

构件就是一个实际文件，可以有以下几种类型：

（1）部署构件（deploymentcomponent），如 dll 文件、exe 文件、COM +对象、CORBA 对象、EJB、动态 Web 页和数据库表等。

（2）工作产品构件（work productcomponent），如源代码文件、数据文件等，这些构件可以用来产生部署构件。

（3）执行构件（execution component），也就是系统执行后得到的构件。

构件图可以对以下几个方面建模：

（1）对源代码文件之间的相互关系建模，如图 6-15 所示。

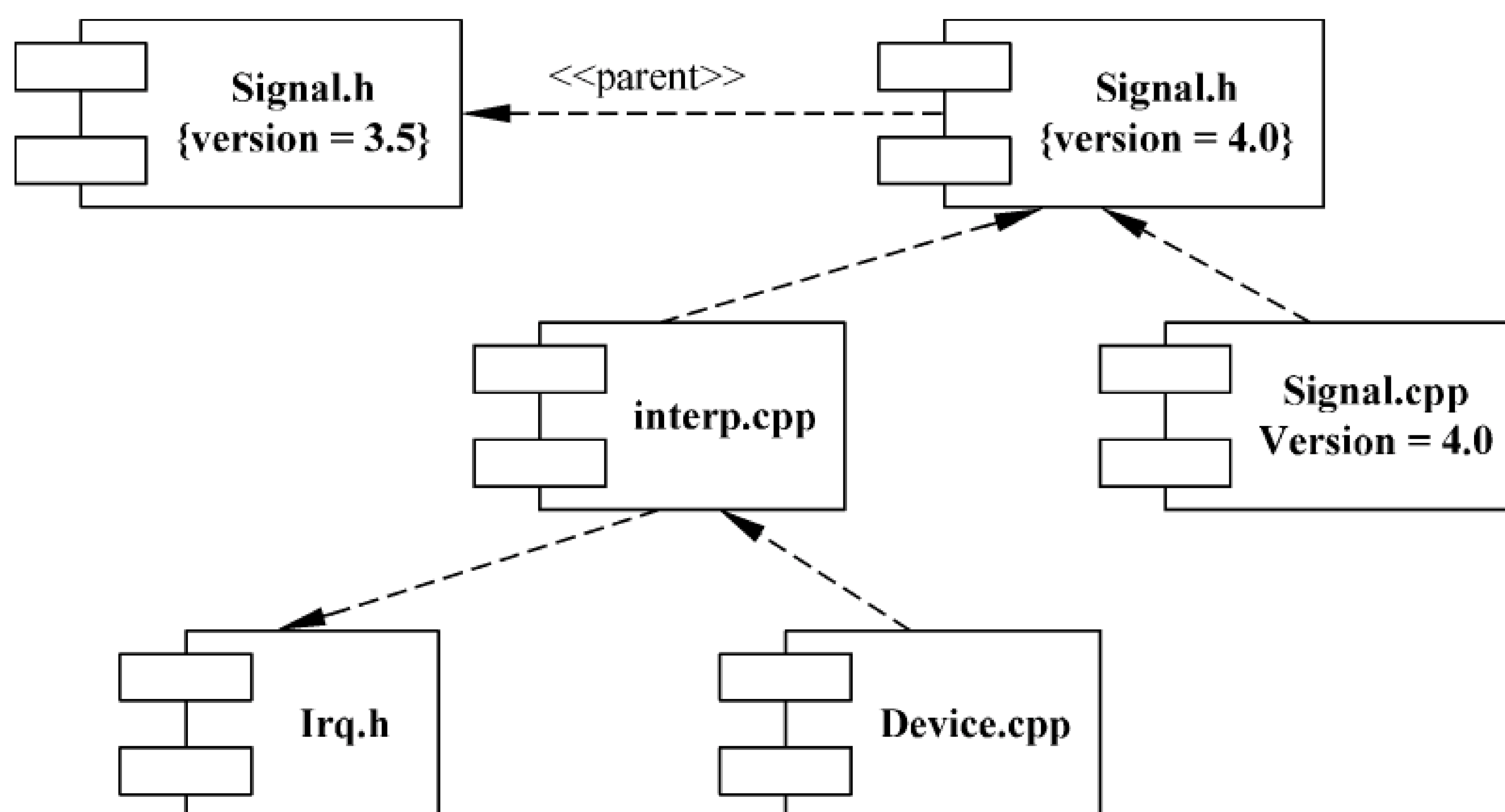


图 6-15　构件图用于对源代码建模

（2）对可执行文件之间的相互关系建模。图 6-16 所示是某可运行系统的部分文件之间的相互关系。

在图 6-16 中，IDriver 是接口，构件 path.dll 和接口 IDriver 之间是依赖关系，而构件 dirver.dll 和接口 IDriver 之间是实现关系。

6.2.7 部署图

部署图也称配置图、实施图，它可以用来显示系统中计算结点的拓扑结构和通信路径与结点上运行的软构件等。一个系统模型只有一个部署图，部署图常用于帮助理解分布式系统。

部署图由体系结构设计师、网络工程师、系统工程师等描述。图6-17所示是一个部署图的例子。

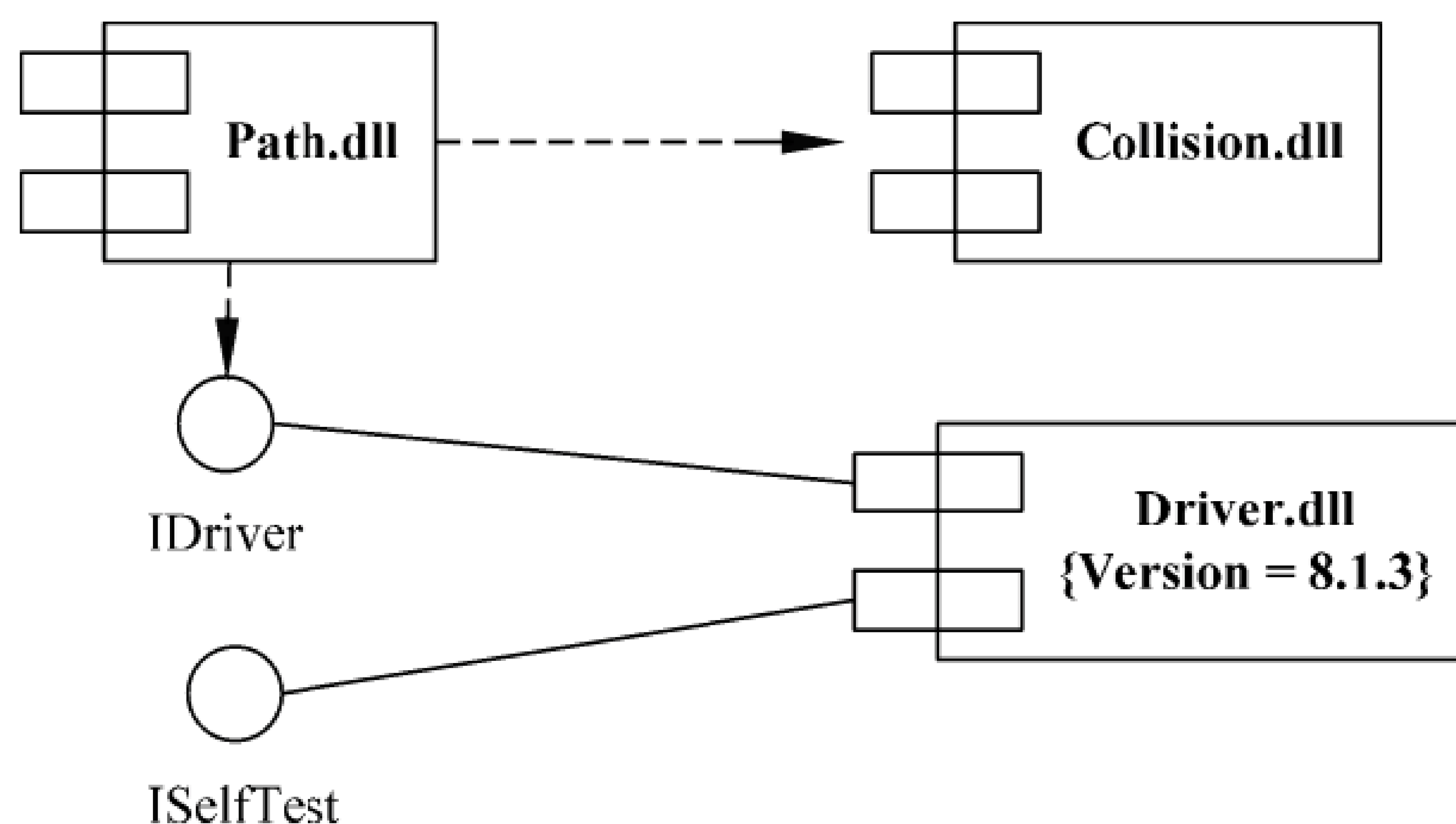


图 6-16 构件图用于对可运行系统建模

图 6-17 部署图

6.3　基于 UML 的软件开发过程

6.3.1　开发过程概述

UML 是独立于软件开发过程的，即 UML 能够在几乎任何一种软件开发过程中使用。迭代的渐进式软件开发过程包含 4 个阶段，即初启、细化、构建和部署。

1．初启

在初启阶段，软件项目的发起人确定项目的主要目标和范围，并进行初步的可行性分析和经济效益分析。

2．细化

细化阶段的开始标志着项目的正式确立。软件项目组在此阶段需要完成以下工作：

（1）初步的需求分析。采用 UML 的用例描述目标软件系统所有比较重要、比较有风险的用例，利用用例图表示参与者与用例以及用例与用例之间的关系。采用 UML 的类图表示目标软件系统所基于的应用领域中的概念与概念之间的关系。这些相互关联的概念构成领域模型。领域模型一方面可以帮助软件项目组理解业务背景，与业务专家进行有效沟通；另一方面，随着软件开发阶段的不断推进，领域模型将成为软件结构的主要基础。如果领域中含有明显的流程处理成分，可以考虑利用 UML 的活动图来刻画领域中的工作流，并标识业务流程中的并发、同步等特征。

（2）初步的高层设计。如果目标软件系统的规模比较庞大，那么经初步需求分析获得的用例和类将会非常多。此时，可以考虑根据用例、类在业务领域中的关系，或者根据业务领域中某种有意义的分类方法将整个软件系统划分为若干个包，利用 UML 的包图刻画这些包及其间的关系。这样，用例、用例图、类、类图将依据包的划分方法分属于不同的包，从而得到整个目标软件系统的高层结构。

（3）部分的详细设计。对于系统中某些重要的或者风险比较高的用例，可以采用交互图进一步探讨其内部实现过程。同样，对于系统中的关键类，也可以详细研究其属性和操作，并在 UML 类图中加以表现。因此，这里倡导的软件开发过程是根据软件元素（用例、类等）的重要性和风险程度确立优先细化原则，建议软件项目组优先考虑重要的、比较有风险的用例和类，不能将风险的识别和解决延迟到细化阶段之后。

（4）部分的原型构造。在许多情形下，针对某些复杂的用例构造可实际运行的原型是降低技术风险、让用户帮助软件项目组确认用户需求的最有效的方法。为了构造原型，需要针对用例生成详尽的交互图，对所有相关类给出明确的属性和操作定义。

综上所述，在细化阶段可能需要使用的 UML 语言机制包括描述用户需求的用例及用例图、表示领域概念模型的类图、表示业务流程处理的活动图、表示系统高层结构的包图和表示用例内部实现过程的交互图。

3．构建

在构造阶段，开发人员通过一系列的迭代完成对所有用例的软件实现工作，在每次迭代中实现一部分用例。以迭代方式实现所有用例的好处在于，用户可以及早参与对已实现用例的实际评价，并提出改进意见。这样可有效降低大型软件系统的开发风险。在实际开始构造软件系统之前，有必要预先制定迭代计划。计划的制定需遵循如下两项原则：

（1）用户认为业务价值较大的用例应优先安排。

（2）开发人员评估后认为开发风险较高的用例应优先安排。

在迭代计划中，要确定迭代次数、每次迭代所需时间以及每次迭代中应完成（或部分完成）的用例。

每次迭代过程由针对用例的分析、设计、编码、测试和集成 5 个子阶段构成。在集成之后，用户可以对用例的实现效果进行评价，并提出修改意见。这些修改意见可以在本次迭代过程中立即实现，也可以在下次迭代中再予以考虑。

构建过程中，需要使用 UML 的交互图来设计用例的实现方法。为了与设计得出的交互图协调一致，需要修改或精化在细化阶段绘制的作为领域模型的类图，增加一些为软件实现所必需的类、类的属性或方法。

在构建阶段的每次迭代过程中，可以对细化阶段绘出的包图进行修改或精化，以便包图切实反映目标软件系统最顶层的结构划分状况。

综上所述，在构建阶段可能需要使用的 UML 语言机制包括：

（1）用例及用例图。它们是开发人员在构造阶段进行分析和设计的基础。

（2）类图。在领域概念模型的基础上引进为软件实现所必需的类、属性和方法。

（3）交互图。表示针对用例设计的软件实现方法。

（4）状态图。表示类的对象的状态-事件-响应行为。

（5）活动图。表示复杂的算法过程，尤其是过程中的并发和同步。

（6）包图。表示目标软件系统的顶层结构。

（7）构件图。

（8）部署图。

4．部署

在部署阶段，开发人员将构造阶段获得的软件系统在用户实际工作环境（或接近实际的模拟环境）中试运行，根据用户的修改意见进行少量调整。

6.3.2 基于 UML 的需求分析

在初步的业务需求描述已经形成的前提下，基于 UML 的需求分析过程（见图 6-18）大致可分为以下步骤。

- 利用用例及用例图表示需求。从业务需求描述出发获取执行者和场景；对场景进

行汇总、分类、抽象，形成用例；确定执行者与用例、用例与用例图之间的关系，生成用例图。

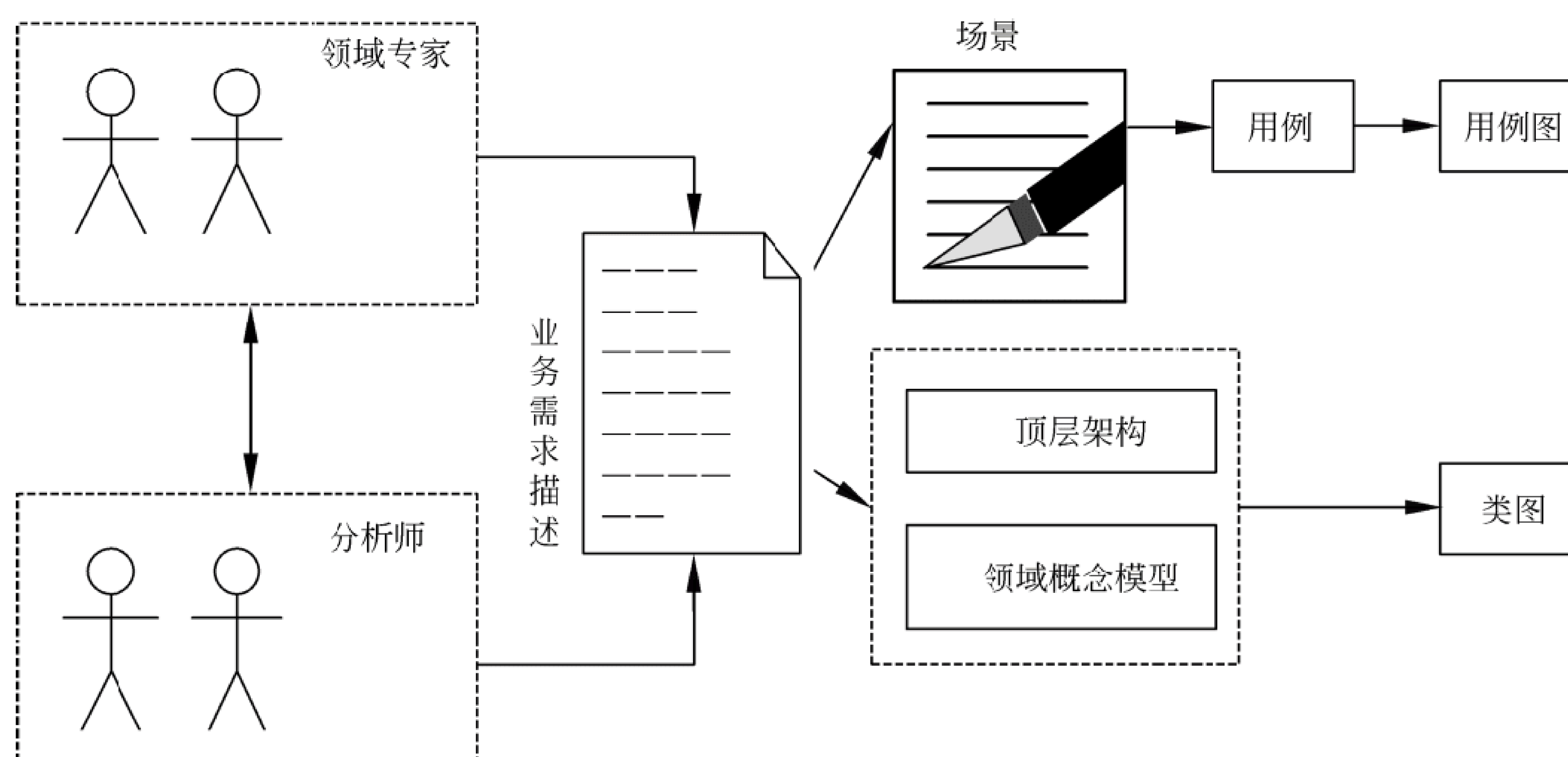


图 6-18　需求分析过程

- 利用包图及类图表示目标软件系统的总体框架结构。根据领域知识、业务需求描述和既往经验设计目标软件系统的顶层架构；从业务需求描述中提取"关键概念"，形成领域概念模型；从概念模型和用例出发，研究系统中主要的类之间的关系，生成类图。

上述两个步骤并没有时序关系，它们可以并行展开。

1．生成用例

从外部用户的视角看，一个用例是执行者（actor）与目标软件系统之间的一次典型的交互作用。从软件系统内部的视角出发，一个用例代表系统执行的一系列动作，动作执行的结果能够被外部的执行者所察觉。执行者是指外部用户或外部实体在系统中扮演的角色。如果多个用户在使用目标软件系统时扮演同一角色，这些用户将由单一执行者表示。反之，如果一个用户扮演多种角色，则需要用多个执行者来表示同一用户。

对用例的完整描述包括用例名称、参与执行者、前置条件、一个主事件流、零到多个辅事件流和后置条件。主事件流表示正常情况下执行者与系统之间的信息交互及动作序列，辅事件流则表示特殊情况或异常情况下的信息交互及动作序列。显式地分隔主、辅事件流是为了使分析人员首先聚焦于正常的业务处理流程，同时也便于用例的读者理解业务需求。

用例主要来源于分析人员对场景的分类和抽象，即将相似的场景进行归并，使一个用例可以通过实例化和参数调节而涵盖多个场景。

例如，在"家庭保安系统"中，执行者有"用户"、"传感器"、"警报器"、"报警电

话”和“显示器”，用例有“系统配置”、“命令响应”和“传感器监测”。下面以“传感器监测”为例说明用例的一般描述格式。

用例名称：传感器监测。

参与执行者：各类传感器、警报器、报警电话和显示器。

前置条件：系统已开机。

主事件流：

①传感器向目标软件系统上报其监测数据，系统判别监测数据是否正常。

②如果不正常，系统启动警报器，拨报警电话号码。

③报警电话接通后，软件系统播出语音，报告异常事件发生的时间、地点和事件的性质。

④系统在控制面板的显示器上显示报警时间及当前状态（报警）。

辅事件流：

①如果报警电话无人接听，则按照重拨延迟反复拨号，直至电话接通，再转入主事件流的步骤③。

②如果重拨次数达到系统预设的最大次数，电话仍无人接听，则跳过主事件流的步骤③，转入步骤④。

后置条件：如果已发现异常的监测数据，系统处于“报警”状态；否则，系统处于正常的“监测”状态。

2．用活动图表示用例

针对前面所述的“传感器监测”用例，其活动图表示如图 6-19 所示。

3．生成用例图

执行者与用例之间的关系有两种：触发执行与信息交换。执行者与用例之间可能兼具这两种关系，例如，在“家庭保安系统”中，执行者“用户”在触发用例“命令响应”的同时，还要向用例传送命令信息。

在 UML 用例图中，从执行者指向用例的边表示触发执行和 / 或信息交换，从用例指向执行者的边则表示用例将其生成的信息传递给执行者。例如图 6-19 中的“传感器监测”用例仅包含正常的处理流程，而“报警电话未接通”用例除正常流程外还增加了“重复拨号”以及“重拨次数达到最大次数仍无人接听”这两种异常处理动作。

4．建立顶层架构

顶层架构的主要目的是为后续的分析和设计活动建立一种结构和分划，以便开发人员在不同的开发阶段，以及同一开发阶段的不同开发人员，能够聚焦于系统的不同部分。顶层架构是分析和设计的阶段成果的承载体。随着开发过程的推进，框架中的内容不断丰富、翔实，最终演进为完整的面向对象软件结构。

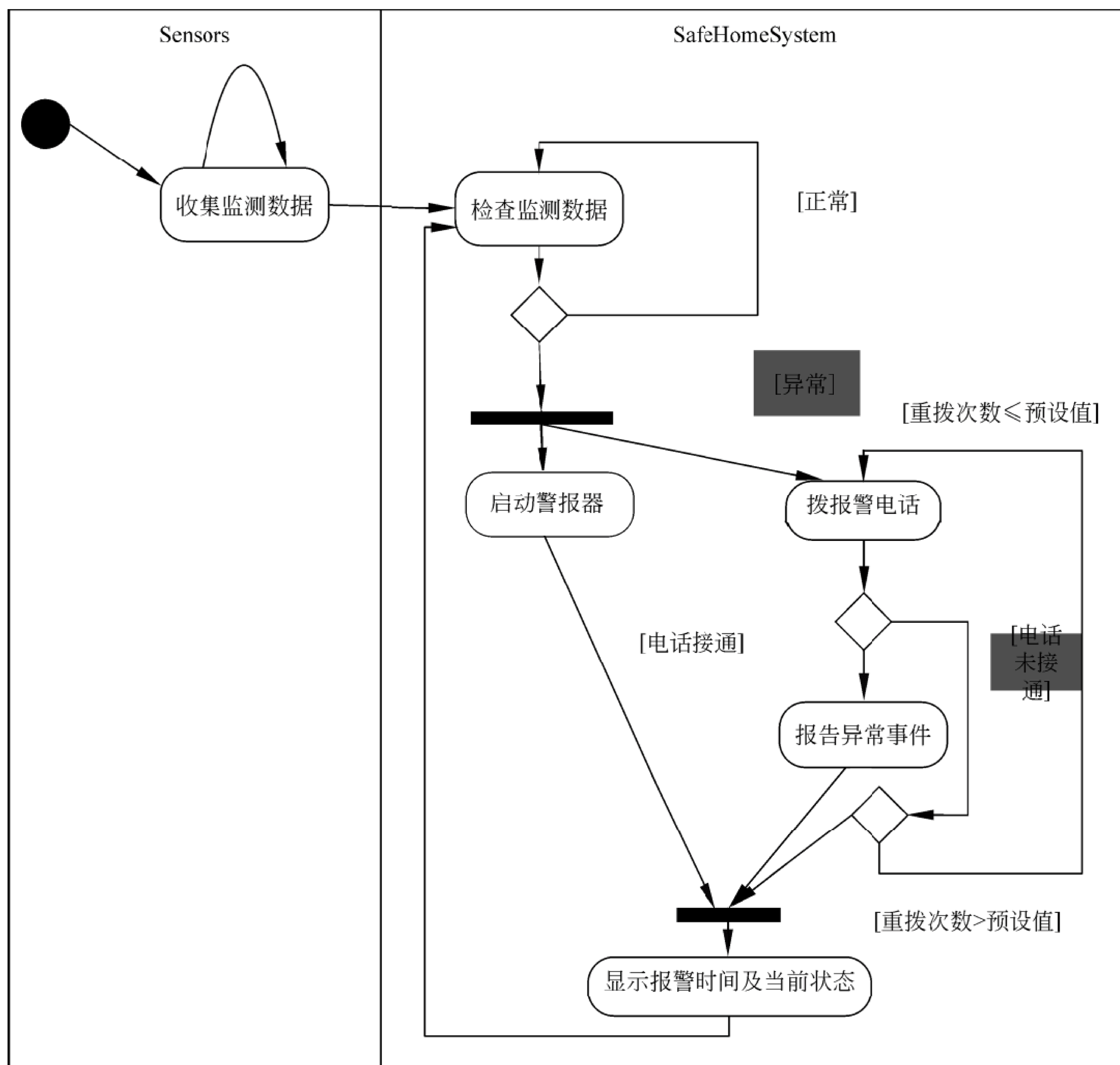


图 6-19　活动图

1）UML 包图

包是 UML 对类进行分组的一种机制。可以从某种视角将具有比较密切的关联的一些类划分为一个包，分属于不同包的两个类之间的关联则比较松散。由此可见，对于大型软件系统而言，包的划分是实现“分而治之”的重要技术手段。

包之间存在两种依赖关系：依赖和构成。如果对类 A 的修改将导致类 B 的改变，则称 B 依赖于 A。如果两个包中存在具有依赖关系的两个类，则认为这两个类分属的包之间存在依赖关系。

2）顶层架构设计

软件系统顶层架构的基本方法是，结合实际需求，从既往的架构设计经验模式中选

取适当者，再进行微调或局部改造。目前有如下几种主要的架构模式：

（1）流程处理模式。流程处理系统以算法和数据结构为中心，其系统功能由一系列的处理步骤构成，相邻的处理步骤之间以数据流通管道相互连接。

（2）客户/服务器模式。客户端负责用户输入和处理结果的呈现，服务器端则负责后台的业务逻辑处理。

- 模型——视图——控制器（Model、View、Controller，MVC）模式。该模式将整个软件系统划分为模型、视图和控制器三个部分。模型负责维护并保存具有持久性的业务数据，实现业务处理功能，并将业务数据的变化情况及时通知视图；视图负责呈现模型中包含的业务数据，响应模型变化通知，更新呈现形式，并向控制器传递用户的界面动作；控制器负责将用户的界面动作映射为模型中的业务处理功能并实际调用之，然后根据模型返回的业务处理结果选择新的视图。MVC模式特别适合于分布式应用软件，尤其是 Web 应用系统。
- 分层模式。分层模式将整个软件系统分为若干层次，最顶层直接面向用户提供软件系统的操作界面，其余各层为紧邻其上的层次提供服务。分层模式可以有效地降低软件系统的耦合度，因此其应用十分普遍。

事实上，大型软件的顶层架构往往需要复合使用多种架构样式。例如，整个目标软件系统采用分层结构，在系统的不同层次内再分别使用适宜的其他种类的架构模式。

在确立顶层架构的过程中需综合考虑以下因素：

- 架构中包的数量。原则上，如果每个包中包含的软件元素（例如类）的数量过多，应考虑将其进一步细分；如果过少，则说明架构过早地陷入了细节，架构划分返工的可能性较大，同时也不合理地限制了后续分析和设计活动的自由空间。
- 架构中包之间的耦合度。包之间的依赖关系和连接关系应尽量简单、稀疏。
- 软件系统的稳定性。要尽量抽取不稳定的软件元素之中相对稳定的部分，将不稳引起的软件元素分类聚集于少数几个包中，以提高软件系统的可维护性。
- 软件系统的必然性。可以将可选功能和必须实现的功能分置于架构中不同的包或子包之中。
- 作为软件系统运行环境的物理网络拓扑。根据软件元素在分布环境中的部署情况。区分顶层架构中的包，可以使包之间的消息传递与物理节点之间的通信相吻合，使后续的分析和设计活动受益于顶层架构中明确定义的通信关系。
- 软件元素的安全、保密级别。根据安全访问的权限划分顶层架构中的包或者子包。
- 开发团队的技术专长。根据开发人员在问题领域和软件技术领域不同的专长划分顶层架构中的包，使每个包都能分配给最适合的开发人员进行后续的分析、设计、编码和测试等，从而有利于并行开发。

5．建立概念模型

例如，“家庭保安系统”的领域概念模型如图 6-20 所示。

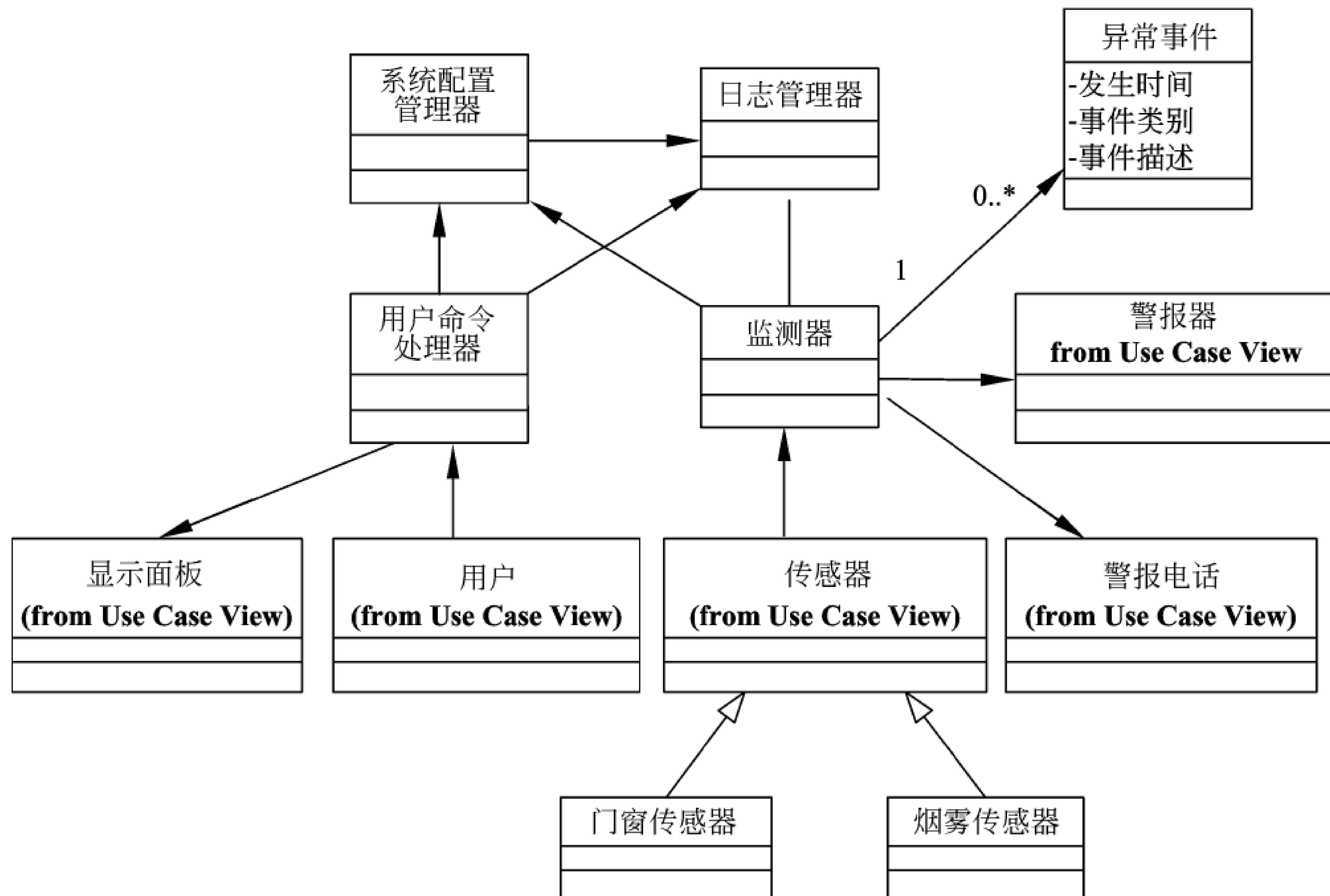


图 6-20　“家庭保安系统”的领域概念模型

6.3.3　面向对象的设计方法

面向对象的软件设计过程如图 6-21 所示。

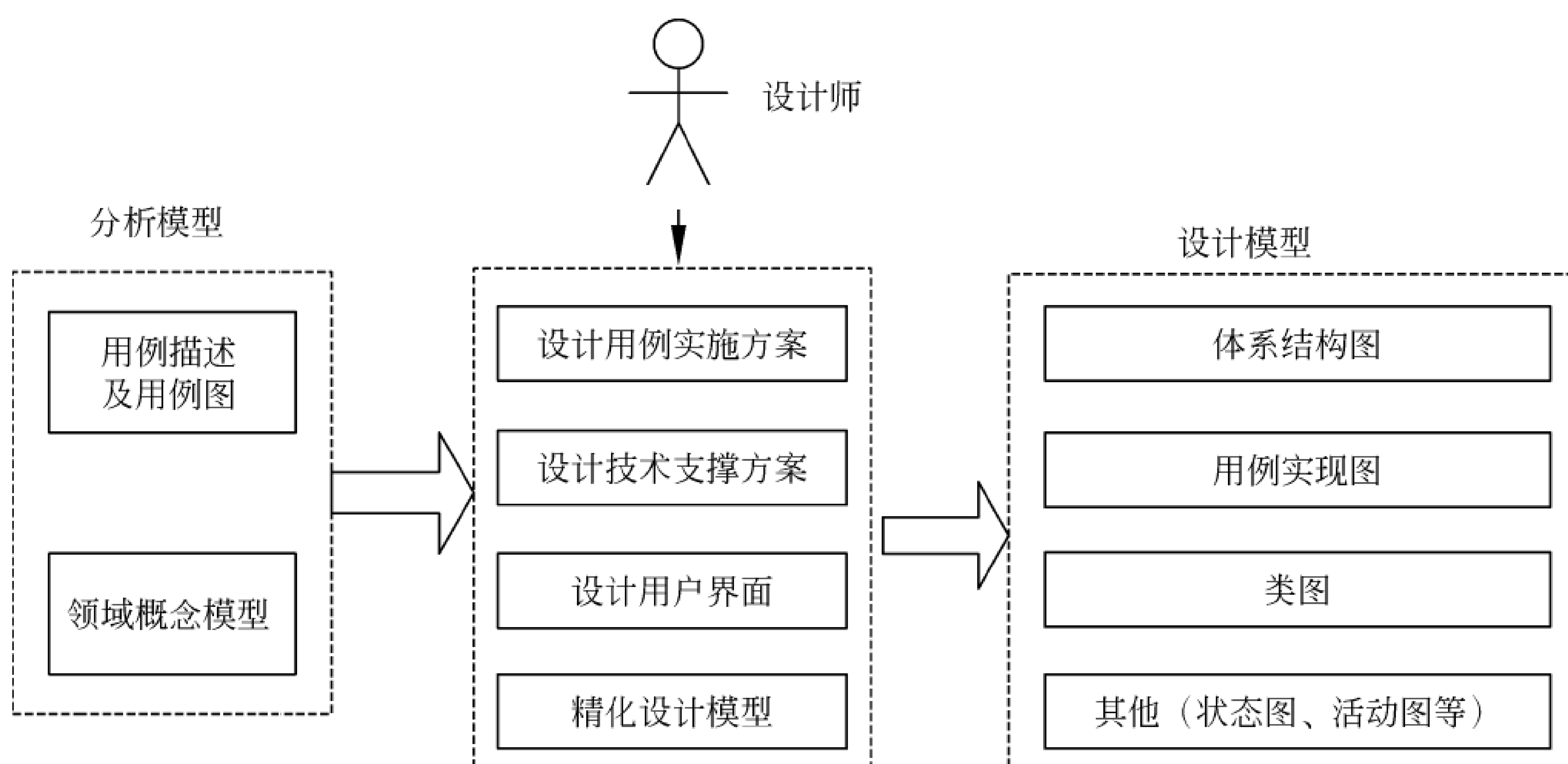


图 6-21　面向对象的软件设计过程

1．设计用例实现方案

UML 的交互图（顺序图、协作图）适于用例实现方案的表示。该设计方法包含如下三个步骤：

（1）提取边界类、实体类和控制类。

边界类用于描述目标软件系统与外部环境之间的交互，并负责实现如下功能：

- 界面控制。包括输入数据的格式及内容转换、输出结果的呈现以及软件运行过程中界面的变化写切换等。
- 外部接口。实现目标软件系统与外部系统或外部设备之间的信息交流和互操作。主要关注跨越目标软件系统边界的通信协议。
- 环境隔离。将目标软件系统与操作系统、数据库管理系统、应用服务器中间件等境软件进行交互的功能与特性封装于边界类之中，使目标软件系统的其余部分尽可能地独立于环境软件。

在 UML 类图中，边界类往往附加 UML 构造型<<boundary>>作为特别标识。

实体类表示目标软件系统中具有持久意义的信息项及其操作。实体类的操作具有“内向收敛”特征，它们仅向目标软件系统的其余部分提供读/写信息项内容的必要的操作接口，并不涉及业务逻辑处理。实体类的 UML 构造型为<<entity>>。

控制类作为完成用例任务的责任承担者，协调、控制其他类共同完成用例规定的功能或行为。对于比较复杂的用例，控制类通常并不处理具体的任务细节，但是它应知道如何分解任务，如何将子任务分派给适当的辅助类，以及如何在辅助类之间进行消息传递和协调。控制类的 UML 构造型为<<control>>。

通常情况下，执行者与用例之间的一种通信连接对应一个边界类。但是，如果两个以上的用例与同一执行者交互，并且这些交互具有共同的行为、完成相同或类似的任务，就可以考虑用同一边界类实现用例与执行者之间的交互。这就意味着边界类的作用范围可以超越单个用例。

（2）构造交互图。

UML 交互图，以交互图作为用例的精确实现方案。

如前所述，用例描述中已包含事件流说明。事件流中的事件应直接对应于交互图中的消息，而事件间的先后关系体现为交互图中的时序，对消息的响应则构成消息接收者的职责。这种职责在后续的设计活动中将被确立为类的方法。

对于比较复杂的用例而言，仅仅依靠控制类、边界类和实体类并不能很好地解决问题，因为我们不能使单个控制类过于庞大和复杂，让它既承担控制、协调的任务，又承担复杂的计算任务。因此，在设计复杂用例的实施方案时，应考虑为控制类设置一些独立的辅助类，让控制类将一些任务委托给辅助类完成。例如，在图 6-20 所示的“家庭保安系统”类图中，“系统配置管理器”和“日志管理器”就是这种意义上的辅助类。

在 UML 顺序图中，用例的主动执行者应位于最左侧，紧邻其右的类是作为用户界

面的边界类，再往右是控制类。控制类的右侧应放置辅助类和实体类，它们的右侧是作为外部接口和环境隔离层的边界类，最右侧是位于目标软件系统边界之外的被动执行者。如此布局之后，在顺序图中不应该出现穿越控制类生命线的消息，即主动执行者向边界类发出命令，边界类将命令进行适当转换后传送至控制类，控制类通过消息请求辅助类、实体类的帮助，协调、控制它们共同完成来自主动执行者的命令。在此过程中，控制类或辅助类可以向右侧的边界类发送消息，将信息或外部处理请求由边界类传向外部系统（被动执行者）。按照上述布局规则绘制的典型的顺序图如图 6-22 所示。

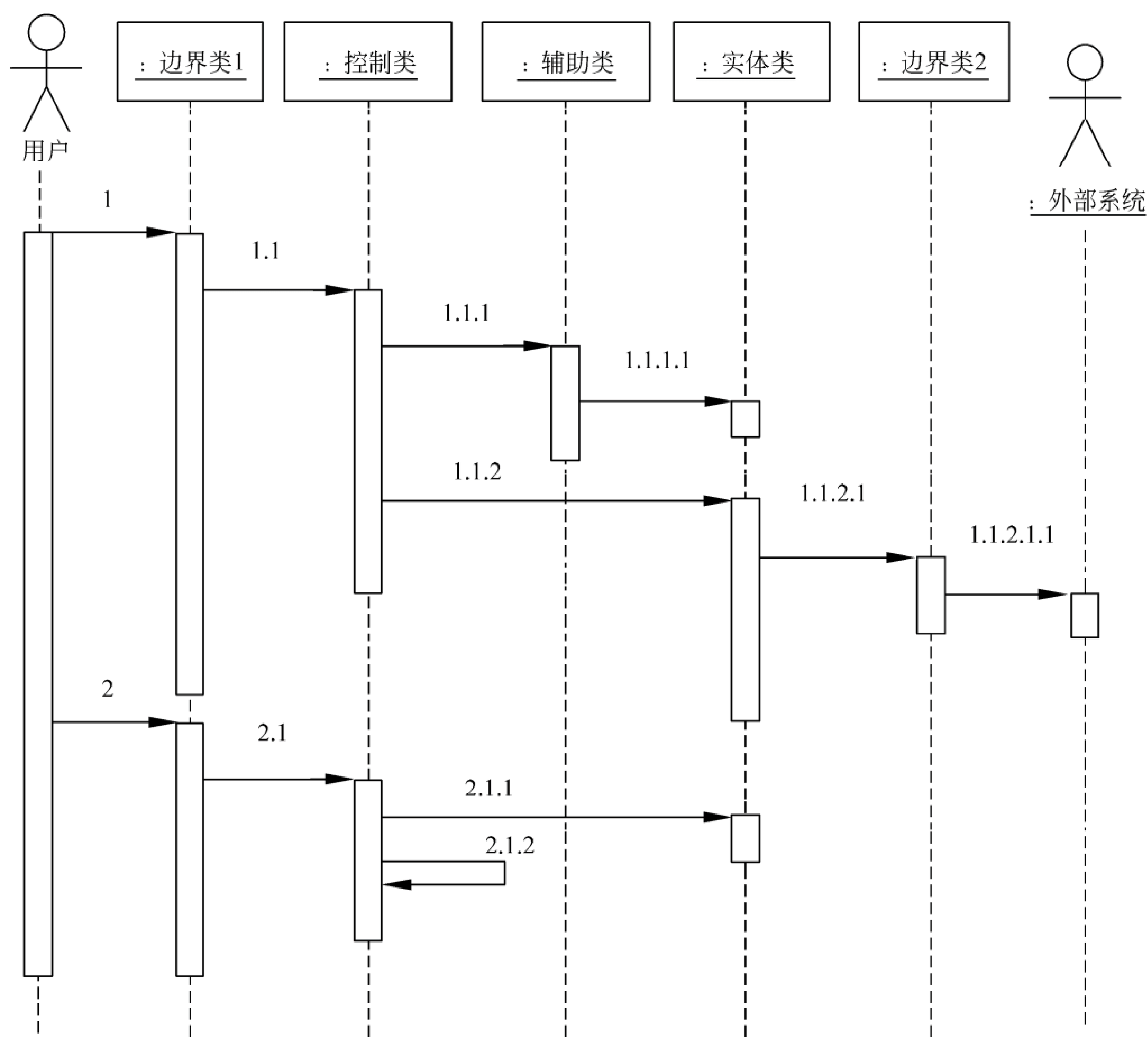


图 6-22　典型布局规则下顺序图

在用例描述中，许多用例除主事件流外，往往还包含备选事件流，以说明在某些特殊或异常情况下的事件和响应动作序列。为易于理解，在设计模型中应该用分离的 UML 交互图分别表示事件流和每个备选事件流。

（3）根据交互图精化类图。

在 UML 交互图中，对每个类的对象都规定了它必须响应的消息以及类的对象之间

的消息传递通道。前者对应于类的操作，后者则对应于类之间的连接关系。因此，可以利用交互图精化分析模型中的类图，将交互图中出现的新类添加到原有类图中，并且对相关的类进行精化，定义其属性和操作。

原则上，每个类都应该有一个操作来响应交互图中指向其对象的那条消息。但是，这并不意味着消息与操作一定会一一对应，因为类的一个操作可能具有响应多条消息的能力。同理，两个类之间的一条连接关系也可以为多条消息提供传递通道。为了简化设计模型，也为了提高重用程度，设计人员应该尽量使用已有的操作来响应新消息，并尽量使用已存在的连接路径作为消息传递的通道。如果两个类之间存在明确、自然的聚合或组合关系，则可以在类图中直接用相应的 UML 图元符号表示类间的聚合或组成关系，这两个关系均可提供消息传递通道。

接下来讨论如何根据交互图确立类的属性。类的操作完成消息响应责任的能力来源于两方面的知识，一是类本身具有的信息，即类的属性；二是类能够找到的其他类，通过其他类协助其完成消息响应。在综合考虑这两个因素之后，类的操作应该明确哪些子任务可通过消息传递路径委托给其他类完成，哪些子任务必须由自身完成。根据后一种子任务的需要，结合领域和业务知识即可推导出类应具有的属性。

2．设计技术支撑方案

在许多软件项目中，应用功能往往都需要一组技术支撑机制为其提供服务。例如，对分布式应用软件（包括电子商务应用、企业 ERP 系统等）而言，需要数据持久存储服务、安全控制服务、分布式事务管理服务、并发与同步控制服务和可靠消息服务等。这些技术支撑设施并非业务需求的直接组成部分，但形态各异的业务处理功能全都依赖于它们提供的公共技术服务。让每个业务功能的设计者直接面对裸机、基本操作系统或基本网络环境来完成软件实现方案是不可思议的。

技术支撑方案应该为多个用例的软件实现提供技术服务，所以，它应该成为整个目标软件系统中全局性的公共技术平台。当用户需求发生变化时，技术支撑方案应具有良好的稳定性。这就要求软件设计者选用开放性和可扩充性较好的技术支撑方案。如果目标软件系统的顶层架构采用分层方式，那么技术支撑方案应该位于层次结构中的较低层次。

技术支撑方案的设计一方面取决于目标软件系统对公共技术服务的需求，另一方面取决于设计人员对软件技术手段的把握和选取。

3．设计用户界面

用户界面设计的策略与步骤如下：

（1）熟悉用户并对用户分类。设计人员应深入用户环境，考虑用户需要完成的任务、完成这些任务需要什么工具支持以及这些工具对用户是否适用。事实上，不同类型的用户要求也不同，一般可按技术熟练程度、工作性质和访问权限对用户进行分类，以便尽量照顾到所有用户的合理要求，并优先满足某些特权用户。

（2）按用户类别分析用户的工作流程与习惯。在用户分类的基础上，从每类中选取一个用户代表，建立包括下列内容的调查表，并通过对调查结果的分析判断用户对操作界面的需求和喜好。

- 姓名。
- 期望软件用途。
- 特征（如年龄、文化程度、限制等）。
- 主要要求与喜好。
- 技术熟练程度。
- 任务客观场景描述。

（3）设计命令系统并进行优化。在设计一个新命令系统时，应尽量遵循用户界面的一般原则和规范，必要时可参考一些优秀的商品软件。根据用户分析结果确定初步的命令系统，然后再优化。命令系统既可为若干菜单、菜单栏，也可为一组按钮。优化命令系统时首先应考虑命令的顺序，一般常用命令居先，命令的顺序与用户工作习惯保持一致；其次，根据外部服务之间的聚合关系组织相应的命令，总体功能对应父命令，部分功能对应子命令；然后，充分考虑人类记忆的局限性（即所谓“7 12”原则或“3×3”原则），命令系统最好组织为一棵两层的多叉树；最后，应尽可能减少用户完成一个操作所需的动作（如单击鼠标、拖曳鼠标和敲击键盘等），并为熟练用户提供操作的快捷方式。

（4）设计用户界面的各种细节。此步骤包括设计一致的用户界面风格、耗时操作的状态反馈、undo 机制、帮助用户记忆的操作序列和自封闭的集成环境等。

（5）增加用户界面专用的类与对象。用户界面专用类的设计与所选用的图形用户界面（GUI）工具或者支持环境有关。一般而言，需要为窗口、菜单、对话框等界面元素定义相应的类，这些类往往继承自 GUI 工具或者支持环境提供的类库中的父类。最后，还需要针对每个与用户命令处理相关的界面类，定义控制设计模型中的其他类的方法。

利用快速原型演示，改进界面设计。为人机交互部分构造原型，是界面设计的基本技术之一。为用户演示界面原型，让他们直观感受目标软件系统的使用方法，并评判系统是否功能齐全、方便好用。

4．精化设计模型

对模型进行改进的活动可以分为精化和合并两种，一般先从精化开始。首先，由于初始架构模型已经包括了总原则和层结构两部分的内容。现在要做的工作是根据需求和架构原则来划分不同的粗粒度组件。粗粒度组件来源于分析活动中的业务实体。把具有很强相关性业务实体组合起来，形成一个集合。集合内部存在错综复杂的关系，同时集合向外部提供服务接口。这样的集合就称为粗粒度组件。粗粒度组件对外的接口和内部的实现是相区分的。粗粒度组件的形式有很多，Java平台上的 Jar 文件、Windows平台上的 dll 文件，甚至古老的.o 或.a 文件都可以是粗粒度组件的表现形式。设计优秀的粗粒度组件应该只是完成一项功能，这一点是它与子系统的主要区分。一个系统中可能包括会

计子系统、库存管理子系统。但是提供会计粗粒度组件或是库存管理粗粒度组件是没有什么意义的。因为这样的粗粒度组件的范围过于广泛，难以发挥重用的价值。粗粒度组件是可以（可能也是必须）跨越层次的。粗粒度组件拥有持久化的行为，拥有业务逻辑，需要表示层的支持。这样看起来，它所属的轴向和层次的轴向是相互垂直的。粗粒度组件来源于需求。需求阶段产生的需求说明书或是用例模型将是粗粒度组件开发的基础。在拥有了需求工件之后，我们需要对需求进行功能性的划分，将需求分为几个功能组，这样我们基本上就可以得到相应的粗粒度组件了。如果系统比较庞大，可以对功能组再做细分。这取决于粗粒度组件的范围。过小的范围，将会造成粗粒度组件不容易使用，用户需要理解不同的粗粒度组件之间的复杂关系，最后的结果也将包含大量的组件和复杂的逻辑。过大的范围，则会造成粗粒度组件难以重用，导致粗粒度组件称为一个子系统。

假设需要开发一个人力资源管理系统。经过整理，它的需求大致分为如下几个部分。

- 组织结构的设计和管理：包括员工职务管理和员工所属部门的管理。
- 员工资料的管理：包括员工的基本资料和简单的考评资料。
- 日常事务的管理：包括了对员工的考勤管理和工资发放管理。

对于前两项的功能组，建立粗粒度组件是比较合适的。但是对于第三项功能组，由于范围过大，将之分为考勤管理和工资管理。现在我们得到了 4 个粗粒度组件。分别是组织结构组件、员工资料组件、员工考勤组件和员工工资组件。

在得到了粗粒度组件之后，下面的工作分为两个部分：第一个部分是定义不同的粗粒度组件之间的关系。第二个部分是在粗粒度组件的基础上定义业务实体或是定义细粒度组件。

不同的粗粒度组件之间的关系其实就是前文提到的粗粒度组件的外部接口。如果可能，在粗粒度组件之间定义单向的关联（如上图所示）可以有效的减少组件之间的耦合。如果必须要定义双向的关联，请确保关联双方组件之间的一致性。在图 6-23 中，我们可以清晰的看出，组织结构处于最底层，员工资料依赖于组织结构，包括从组织结构中获得员工的所属部门，以及员工职务等信息。而对于考勤、工资组件来说，需要从员工资料中获取必要的信息，也包括了部门和职务两方面的信息。这里有两种关联定义的方法，一种是让考勤组件从组织结构组件中获得部门和职务信息，从员工资料中获得另外的信息，另一种是如图 6-23 一样，考勤组件只从员工资料组件中获得信息，而员工资料组件再使用委托，从组织结构中获得部门和职务的信息。第二种做法的好处是向考勤、工资组件屏蔽了组织结构组件的存在，并保持了信息获取的一致性。这里演示的只是组件之间的简单关系，现实中的关系不可能

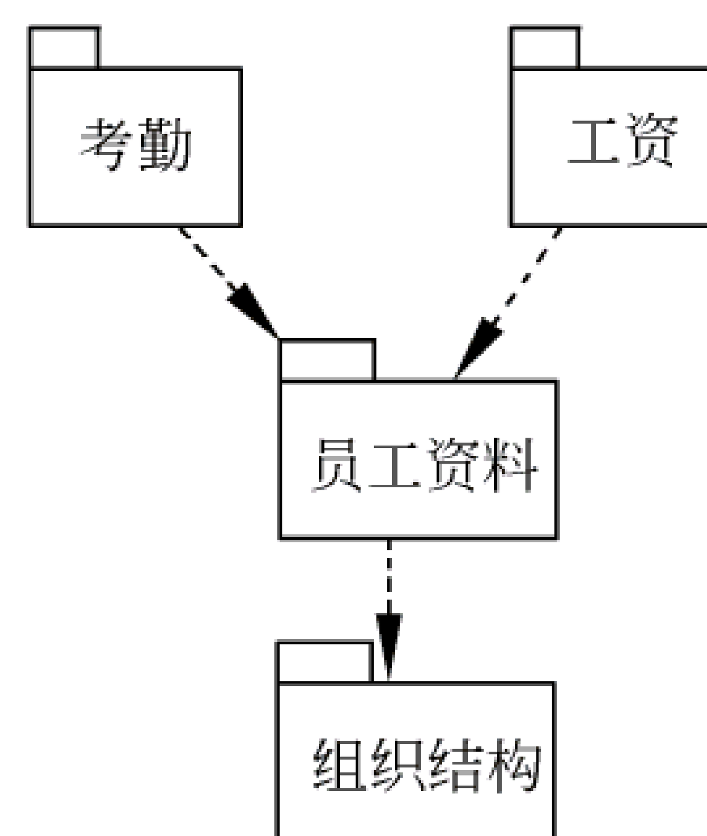


图 6-23 粗粒度组件之间的单向关联

如此的简单，但是处理的基本思路是一样的，就是尽可能简化组件之间的关系，从而减少它们之间的耦合度。

在得到了粗粒度组件模型之后，我们需要对其进行进一步的分析，以得到细粒度的组件。细粒度的组件具有更好的重用性，并使得架构设计的精化工作更进一步。按 Jacobson 推荐的面向对象软件工程（Object Oriented Software Enginerring，OOSE）的做法，我们需要从软件的目标领域中识别出关键性的实体，或者说是领域中的名词。例如上例中的员工、部门、工资等。然后决定它们应该归属于哪些粗粒度组件。先识别细粒度组件还是先识别粗粒度组件并没有固定的顺序。

最初得到的组件模型可能并不完善，需要对其进行修改。可能某个组件中的类太多了，过于复杂，就需要对其进一步精化、分为更细的组件，也许某个组件中的类太少了，需要和其他的组件进行合并。也许你会发现某两个组件之间存在重复的要素，可以从中抽取出共性的部分，形成新的组件。组件分析的过程并没有一种标准的做法，你只能够根据具体的案例来进行分析。

最后的模型将会明确的包含几个经过精化之后的粗粒度组件。粗粒度组件之间的关系也会进行一次重新定义。

6.4　系统架构文档化

6.4.1　模型概述

软件架构用来处理软件高层次结构的设计和实施。它以精心选择的形式将若干结构元素进行装配，从而满足系统主要功能和性能需求，并满足其他非功能性需求，如可靠性、可伸缩性、可移植性和可用性。Perry 和 Wolfe 使用一个精确的公式来表达，该公式由 Boehm 做了进一步修改。

软件架构＝{元素，形式，关系/约束}

软件架构涉及到抽象、分解和组合、风格和美学。我们用由多个视图或视角组成的模型来描述它。为了最终处理大型的、富有挑战性的架构，该模型包含 5 个主要的视图如图 6-24 所示。

- 逻辑视图（logical view），设计的对象模型（使用面向对象的设计方法时）。
- 过程视图（process view），捕捉设计的并发和同步特征。
- 物理视图（physical view），描述了软件到硬件的映射，反映了分布式特性。
- 开发视图（development view），描述了在开发环境中软件的静态组织结构。

架构的描述，即所做的各种决定，可以围绕着这 4 个视图来组织，然后由一些用例（use cases）或场景（scenarios）来说明，从而形成了第 5 个视图。正如将看到的，实际上软件架构部分从这些场景演进而来。

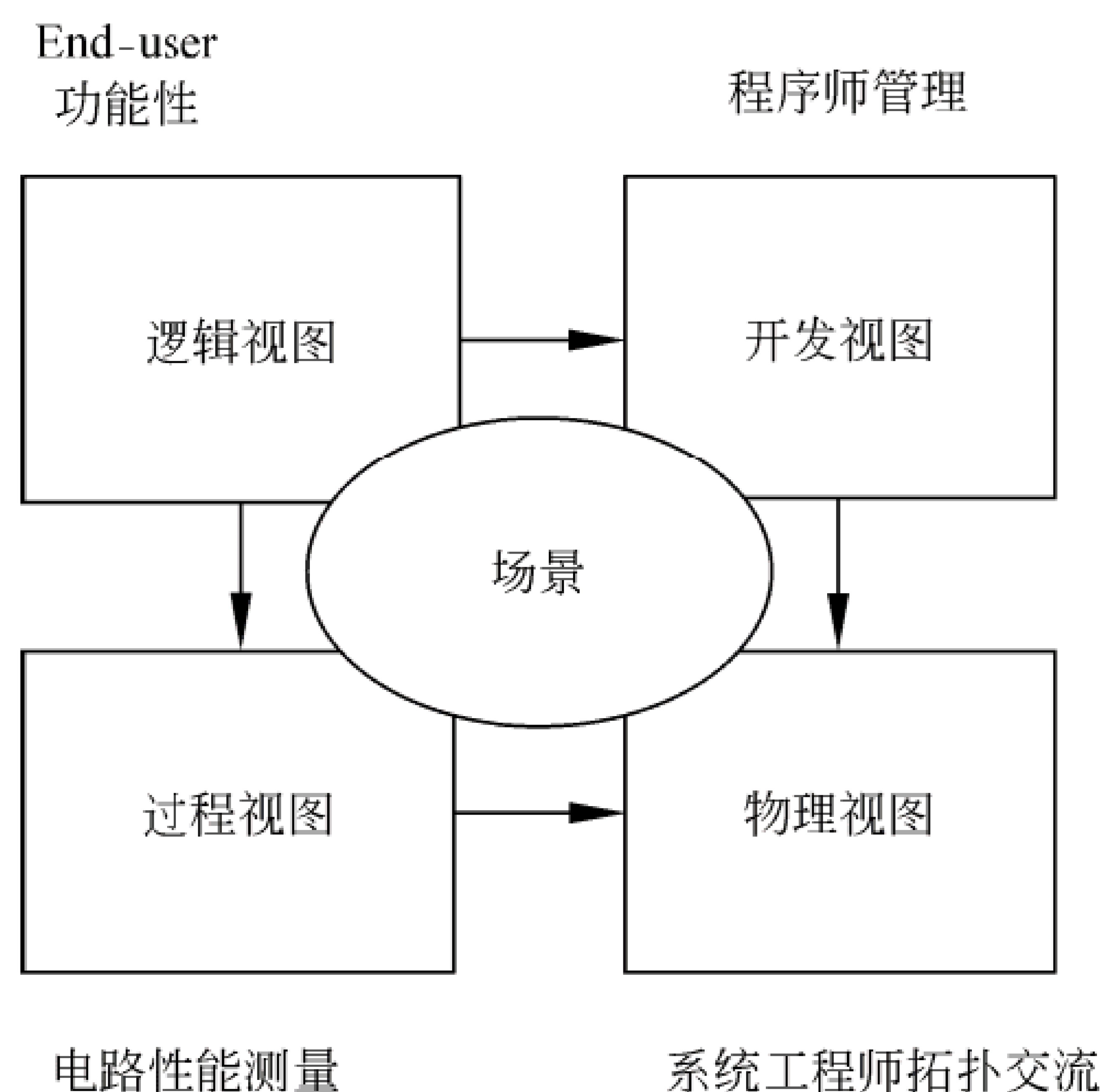


图 6-24　Rational“4+1”视图模型

我们在每个视图上均独立地应用 Perry & Wolf 的公式，即定义一个所使用的元素集合（组件、容器、连接符），捕获工作形式和模式，并且捕获关系及约束，将架构与某些需求连接起来。每种视图使用自身所特有的表示法——蓝图（blueprint）来描述，并且架构师可以对每种视图选用特定的架构风格（architectural style），从而允许系统中多种风格并存。

“4+1”视图模型具有相当的“普遍性”，因此可以使用其他的标注方法和工具，也可以采用其他的设计方法，特别是对于逻辑和过程的分解。

6.4.2　逻辑结构

逻辑架构主要支持功能性需求，即在为用户提供服务方面系统所应该提供的功能。系统分解为一系列的关键抽象，（大多数）来自于问题域，表现为对象或对象类的形式。它们采用抽象、封装和继承的原理。分解并不仅仅是为了功能分析，而且用来识别遍布系统各个部分的通用机制和设计元素。我们使用 Rational/Booch 方法来表示逻辑架构，借助于类图和类模板的手段。类图用来显示一个类的集合和它们的逻辑关系：关联、使用、组合、继承等。相似的类可以划分成类集合，基本的逻辑蓝图表示法如图 6-25 所示。类模板关注于单个类，它们强调主要的类操作，并且识别关键的对象特征。如果需要定义对象的内部行为，则使用状态转换图或状态图来完成。公共机制或服务可以在类功能（class utilities）中定义。对于数据驱动程度高的应用程序，可以使用其他形式的逻辑视图，例如 E-R 图，来代替面向对象的方法。

逻辑视图的表示法来自 Booch 标记法。当仅考虑具有架构意义的条目时，这种表示法相当简单。特别是在这种设计级别上，大量的修饰作用不大。我们使用 Rational Rose 来支持逻辑架构的设计。

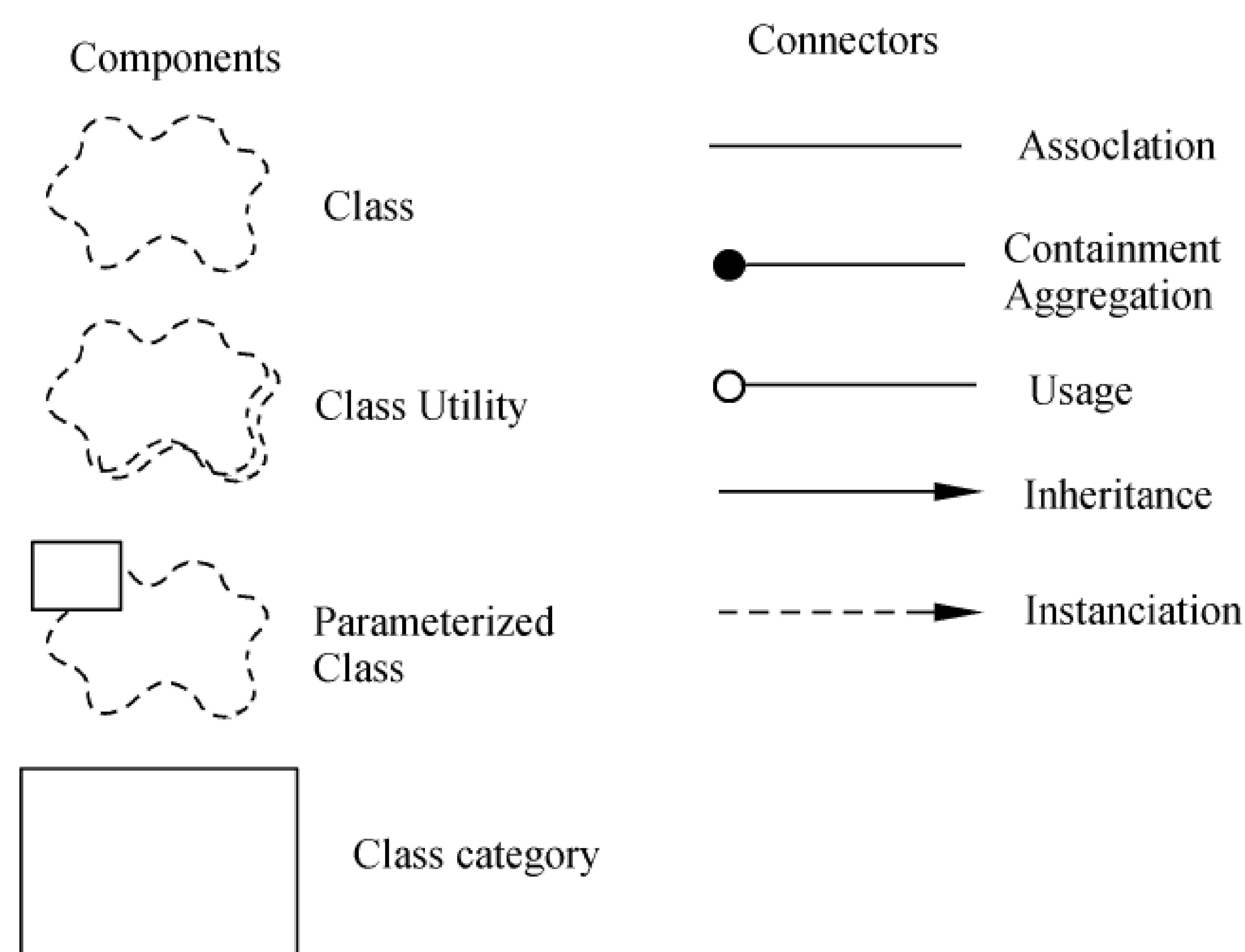


图 6-25　逻辑蓝图的表示法

1．逻辑视图的风格

逻辑视图的风格采用面向对象的风格，其主要的设计准则是试图在整个系统中保持单一的、一致的对象模型，避免就每个场合或过程产生草率的类和机制的技术说明。

2．逻辑结构蓝图的样例

图 6-26 显示了 Télic PABX 架构中主要的类。

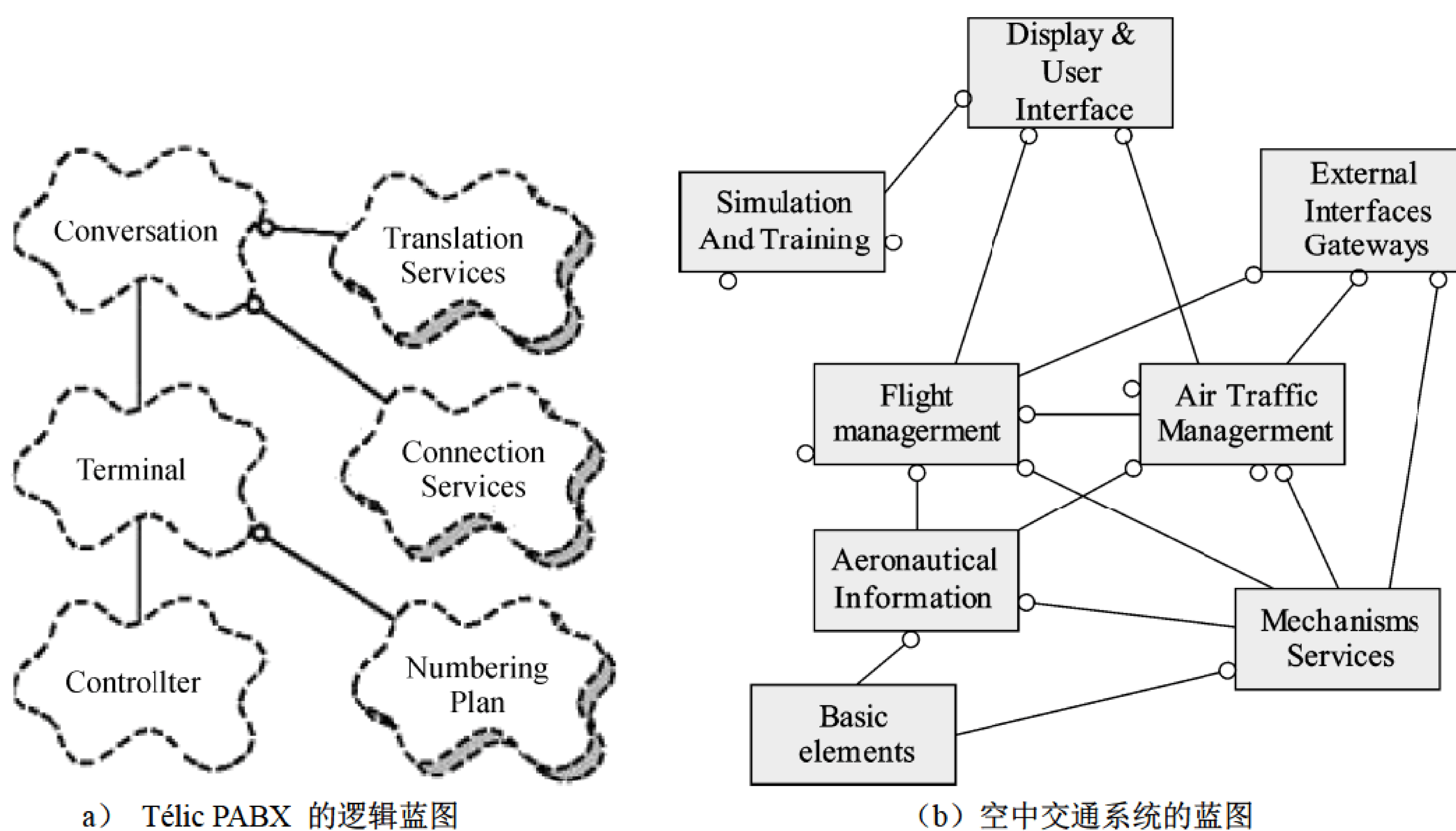


图 6-26　Télic PABX 架构中主要的类

对于一个包含了大量的具有架构重要意义的类的、更大的系统来说，图 6-26（b）描述了空中交通管理系统的顶层类图，包含 8 个类的种类（例如，类的分组）。

6.4.3 进程架构

进程架构考虑一些非功能性的需求，如性能和可用性。它解决并发性、分布性、系统完整性、容错性的问题，以及逻辑视图的主要抽象如何与进程结构相配合在一起——即在哪个控制线程上，对象的操作被实际执行。

进程架构可以在几种层次的抽象上进行描述，每个层次针对不同的问题。在最高的层次上，进程架构可以视为一组独立执行的通信程序（叫作“processes”）的逻辑网络，它们分布在整个一组硬件资源上，这些资源通过 LAN 或者 WAN 连接起来。多个逻辑网络可能同时并存，共享相同的物理资源。例如，独立的逻辑网络可能用于支持离线系统与在线系统的分离，或者支持软件的模拟版本和测试版本的共存。

进程是构成可执行单元任务的分组。进程代表了可以进行策略控制过程架构的层次（即开始、恢复、重新配置及关闭）。另外，进程可以就处理负载的分布式增强或可用性的提高而不断地被重复。

接着，我们可以区别主要任务、次要任务。主要任务是可以唯一处理的架构元素；次要任务是由于实施原因而引入的局部附加任务（如周期性活动、缓冲、暂停等）。它们可以作为 Ada Task 或轻量线程来实施。主要任务的通讯途径是良好定义的交互任务通信机制：基于消息的同步或异步通信服务、远程过程调用、事件广播等。次要任务则以会见或共享内存来通信。在同一过程或处理节点上，主要任务不应对它们的分配做出任何假定。

消息流、过程负载可以基于过程蓝图来进行评估，同样可以使用哑负载来实现“中空”的进程架构，并测量在目标系统上的性能。正如 Filarey etal 在他的 Eurocontrol 实验中描述的那样。

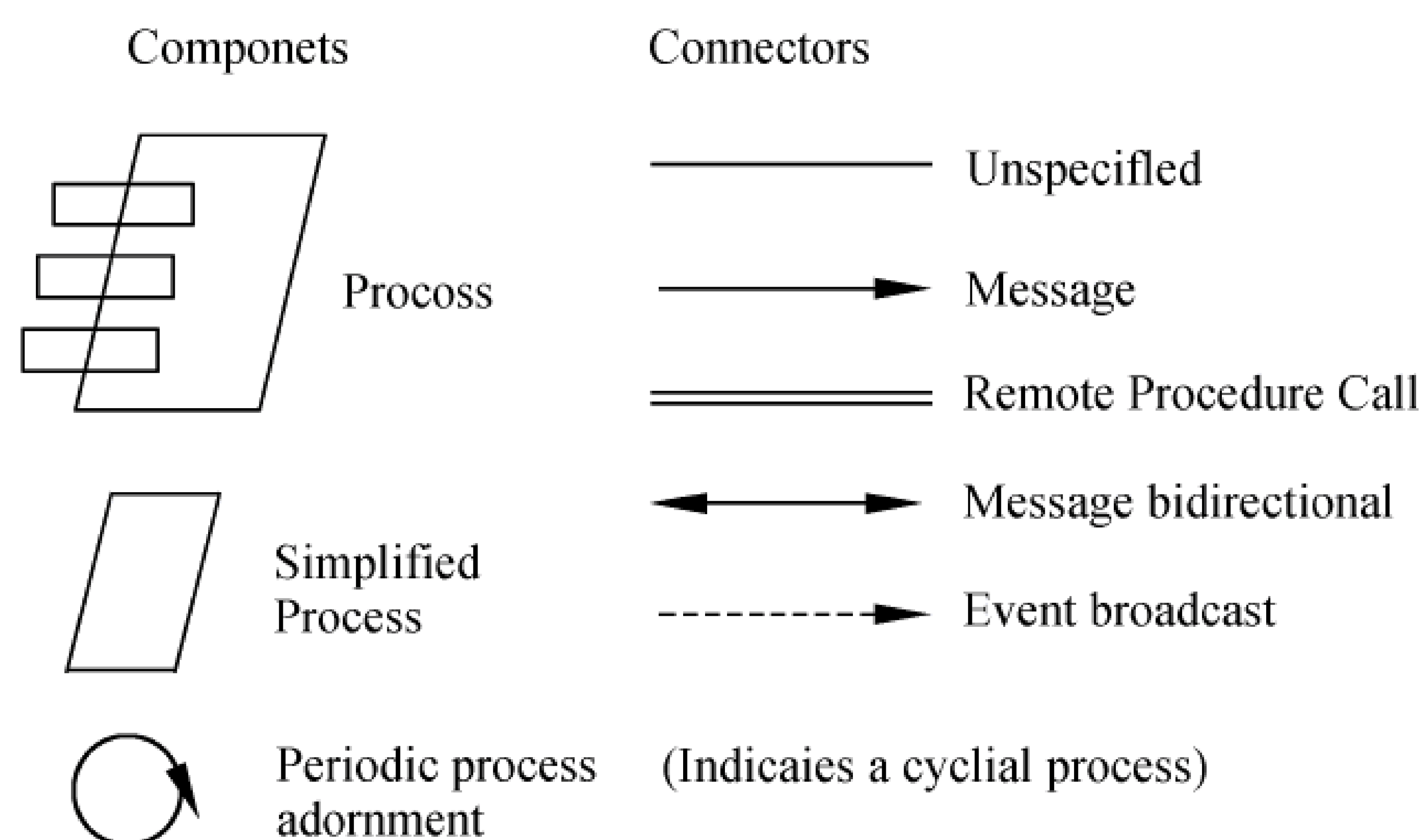


图 6-27　进程蓝图表示法

我们所使用的进程视图的表示方法是从 Booch 最初为 Ada 任务推荐的表示方法扩展而来。同样，用来所使用的表示法关注在架构上具有重要意义的元素，典型的进程蓝图表示法如图 6-27 所示。

许多进程视图的风格可以适用于进程视图。例如采用 Garlan 和 Shaw 的分类法 1，我们可以得到管道和过滤器（pipes and filters），或客户端/服务器，以及各种多个客户端/单个服务器和多个客户端/多个服务器的变体。对于更加复杂的系统，可以采用类似于 K.Birman 所描述的 ISIS 系统中进程组方法以及其他的标注方法和工具。

图 6-28 为进程蓝图的例子。

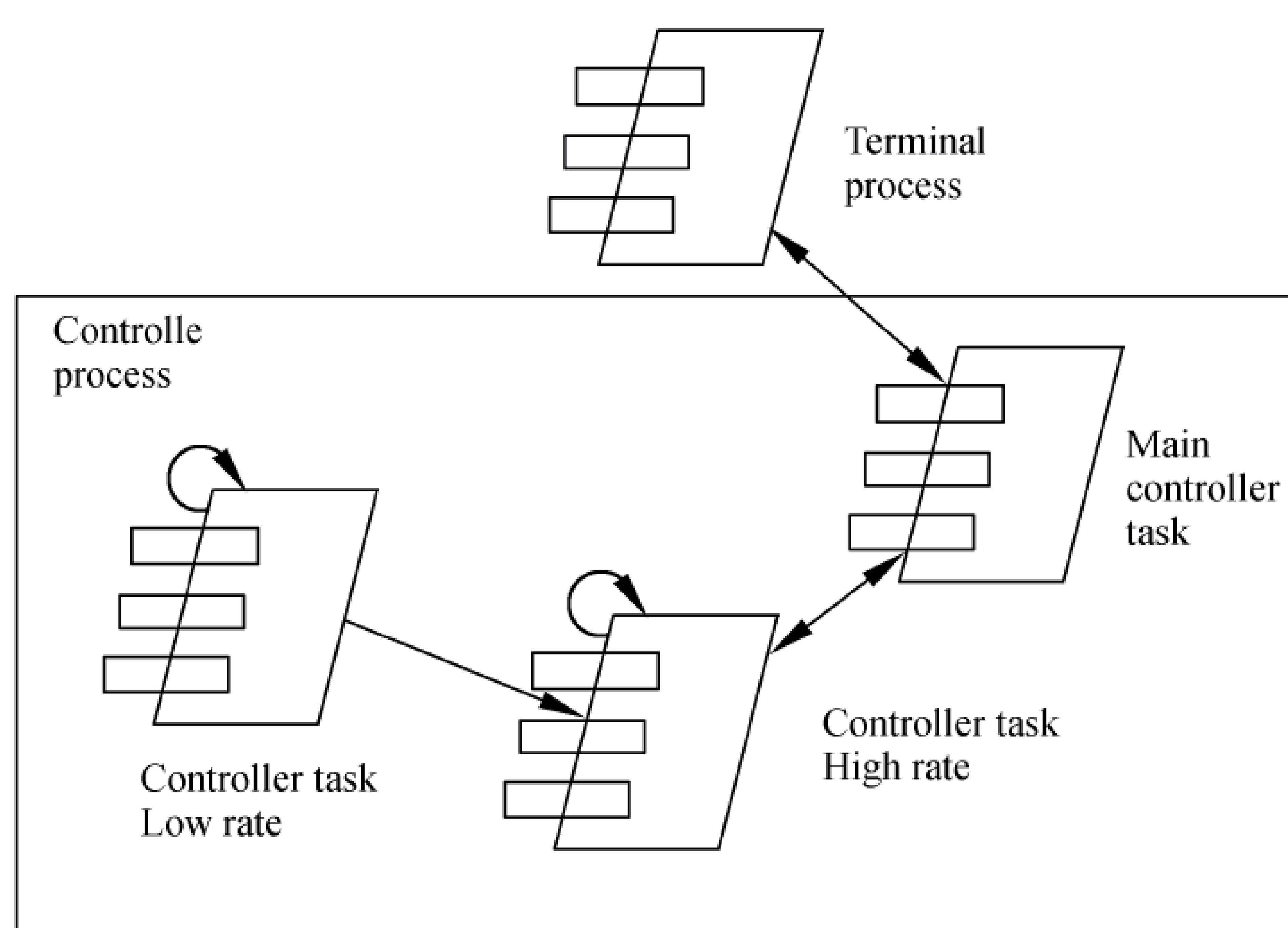


图 6-28　Télic PABX 的进程蓝图（部分）

所有的终端由单个的 Termal process 处理，其中 Termal process 由输入队列中的消息进行驱动。Controller 对象在组成控制过程三个任务之中的一项任务上执行：Low cycle rate task 扫描所有的非活动终端（200 ms），将 High cycle rate task（10 ms）扫描清单中的终端激活，其中 High cycle rate task 检测任何重要的状态变化，将它们传递给 Main controller task，由它来对状态的变更进行解释，并通过向对应的终端发送消息来通信。这里 Controller 过程中的通信通过共享内存来实现。

6.4.4　开发架构

开发架构关注软件开发环境下实际模块的组织。软件打包成小的程序块（程序库或子系统），它们可以由一位或几位开发人员来开发。子系统可以组织成分层结构，每个层为上一层提供良好定义的接口。

系统的开发架构用模块和子系统图来表达，显示了“输出”和“输入”关系。完整的开发架构只有当所有软件元素被识别后才能加以描述。但是，可以列出控制开发架构

的规则：分块、分组和可见性。

大部分情况下，开发架构考虑的内部需求与以下几项因素有关：开发难度、软件管理、重用性和通用性及由工具集、编程语言所带来的限制。开发架构视图是各种活动的基础，如：需求分配、团队工作的分配（或团队机构）、成本评估和计划、项目进度的监控、软件重用性、移植性和安全性。它是建立产品线的基础。

来自 Rational 的 Apex 开发环境支持开发架构的定义和实现和前文描述的分层策略，以及设计规则的实施。Rational Rose 可以在模块和子系统层次上绘制开发蓝图，并支持开发源代码（Ada、C++）进程的正向和反向工程。

关于开发视图的风格，推荐使用分层（layered）的风格，定义 4～6 个子系统层。每层均具有良好定义的职责。设计规则是某层子系统依赖同一层或低一层的子系统，从而最大程度地减少了具有复杂模块依赖关系的网络的开发量，得到层次式的简单策略。

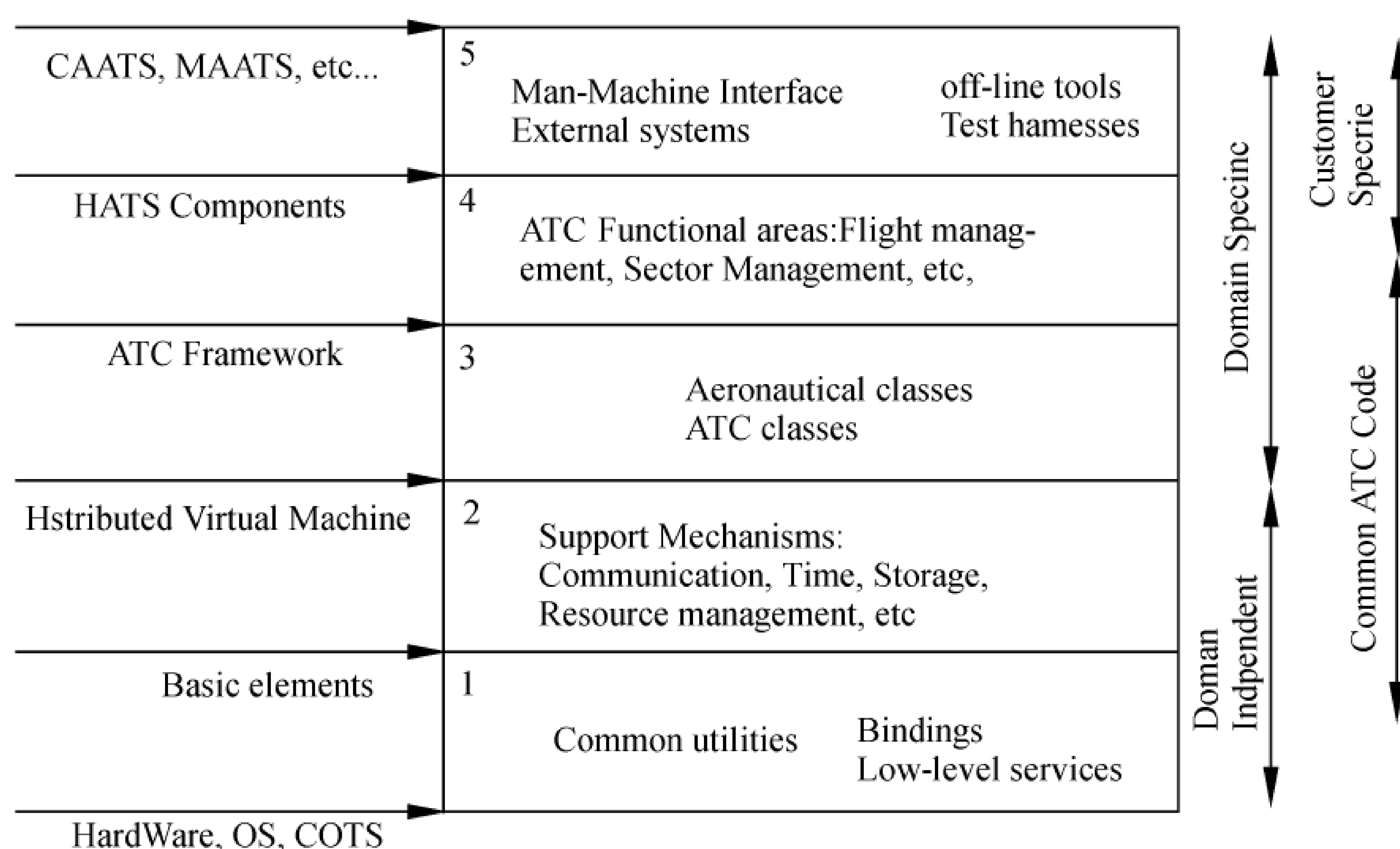


图 6-29　Hughes 空中交通系统（HATS）的 5 个层

图 6-29 代表了加拿大的 Hughes Aircraft 开发的空中交通控制系统（air traffic control system）产品线的 5 个分层开发组织结构。

第一层和第二层组成了独立于域的覆盖整个产品线的分布式基础设施，并保护其免受不同硬件平台、操作系统或市售产品（如数据库管理系统）的影响。第三层为该基础设施增加了 ATC（Adaptive Transform Coding，自适应变换编码方法）框架，形成一个特定领域的软件架构（domain-specific software architecture）。使用该框架，可以在第四层上构建一个功能选择板。层次 5 则非常依赖于客户和产品，包含了大多数用户接口和外部系统接口。72 个子系统分布于 5 个层次上，每层包含了 10～50 个模块，并可以在其他蓝图上表示。

6.4.5　物理架构

物理架构主要关注系统非功能性的需求，如可用性、可靠性（容错性），性能（吞吐量）和可伸缩性。软件在计算机网络或处理节点上运行，被识别的各种元素（网络、过程、任务和对象），需要被映射至不同的节点；我们希望使用不同的物理配置：一些用于开发和测试，另外一些则用于不同地点和不同客户的部署。因此软件至节点的映射需要高度的灵活性及对源代码产生最小的影响。

物理蓝图的表示法，如图 6-30 所示。

大型系统中的物理蓝图会变得非常混乱，所以它们可以采用多种形式，有或者没有来自进程视图的映射均可。

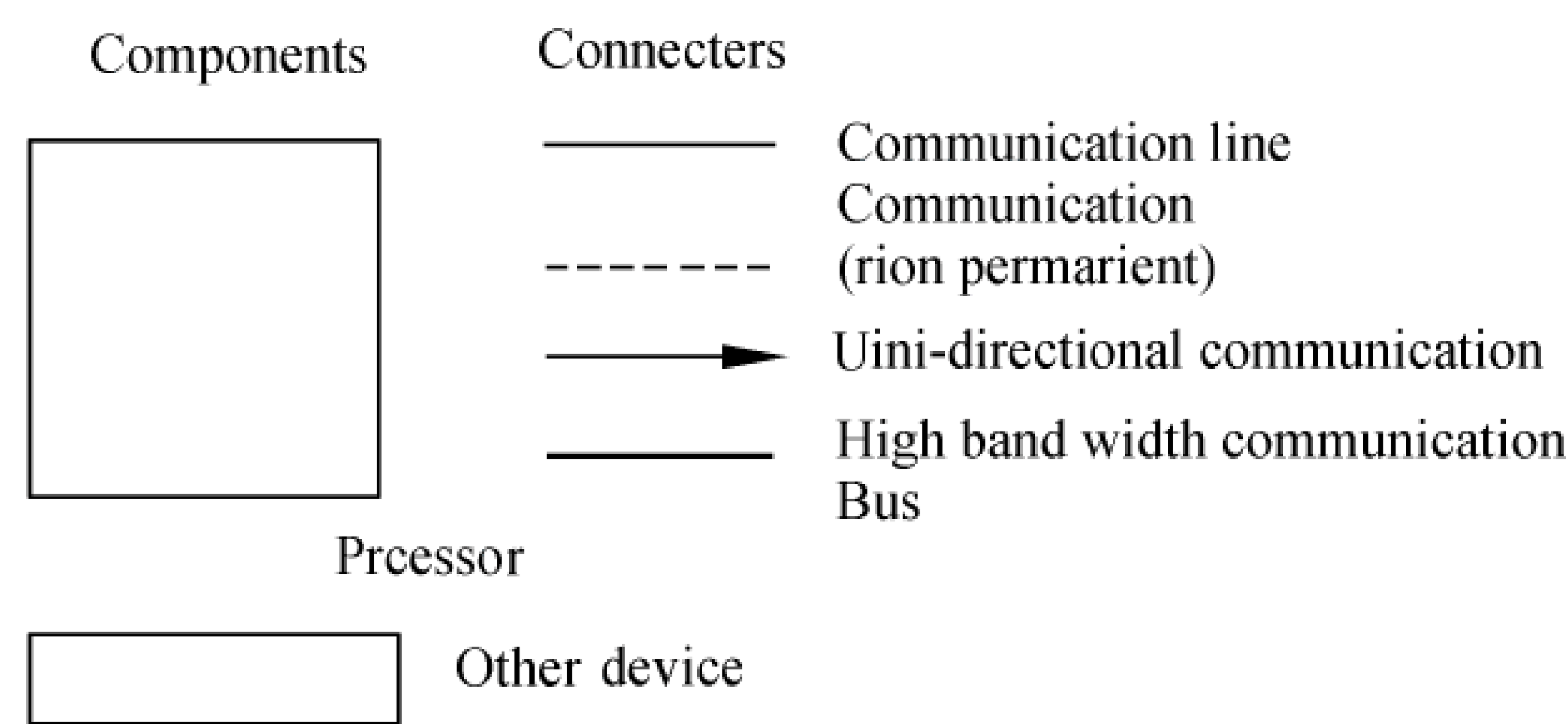


图 6-30　物理蓝图的表示法

TRW（Thompson Ramo Wooldridge Lnc）公司的 UNAS（Universal Network Architecture Services）提供了数据驱动方法将过程架构映射至物理架构，该方法允许大量的映射的变更而无需修改源代码。

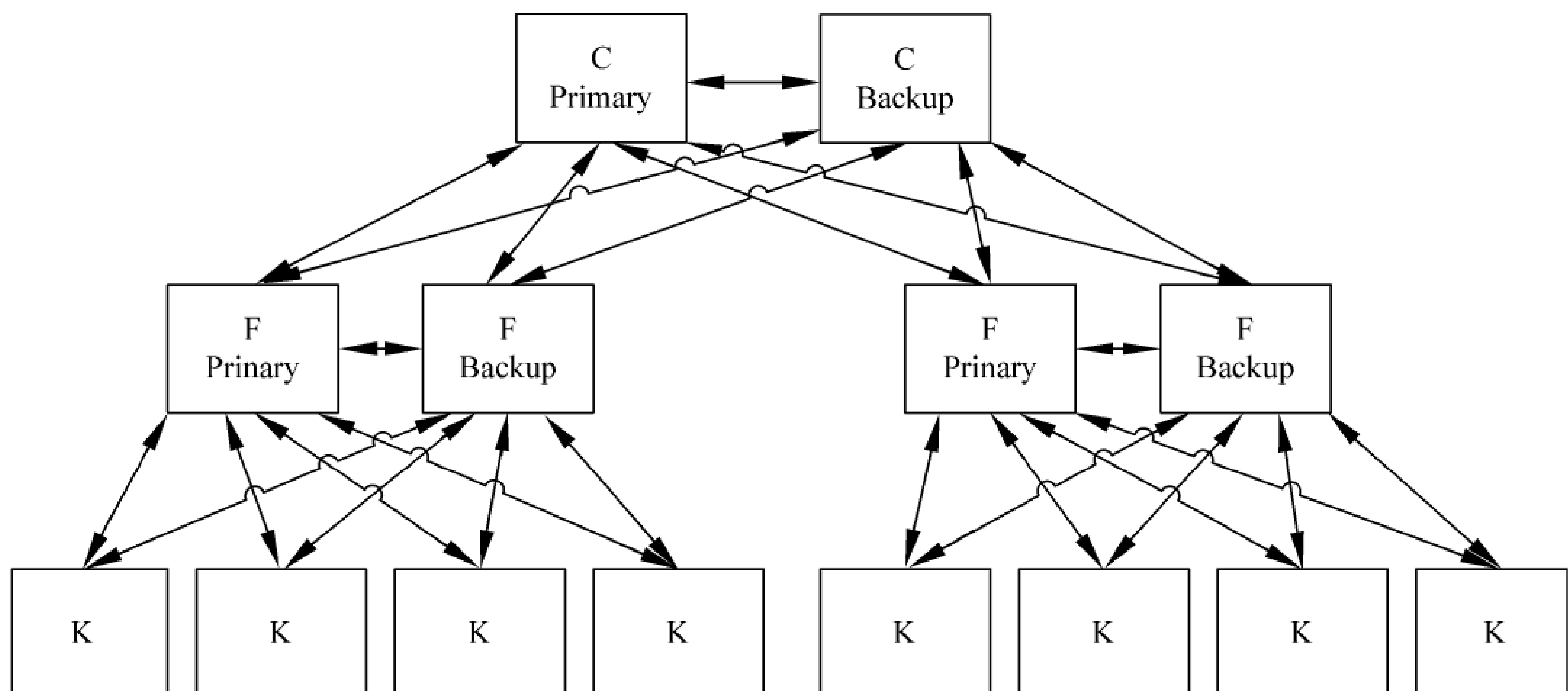


图 6-31　PABX 的物理蓝图

图 6-31 作为物理蓝图的示例，显示了大型 PABX（Private Automatic Branch Exchange）

可能的硬件配置，而图 6-32 和图 6-33 显示了两种不同物理架构上的进程映射，分别对应一个小型和一个大型 PABX。C、F 和 K 是三种不同容量的计算机，支持三种不同的运行要求。

图 6-32 带有过程分配的小型 PABX 物理架构

6.4.6 场景

4 种视图的元素通过数量比较少的一组重要场景（更常见的是用例）进行无缝协同工作，我们为场景描述相应的脚本（对象之间和过程之间的交互序列）。在某种意义上场景是最重要的需求抽象，它们的设计使用对象场景图和对象交互图来表示。

图 6-33 带有过程分配的大型 PABX 物理架构

该视图是其他视图的冗余（因此“+1”），但它起到了两个作用：

- 作为一项驱动因素来发现架构设计过程中的架构元素。
- 作为架构设计结束后的一项验证和说明功能，既以视图的角度来说明又作为架构原型测试的出发点。

场景表示法与组件逻辑视图非常相似（请对照图 6-22），但它使用过程视图的连接符来表示对象之间的交互（请对照图 6-25），注意对象实例使用实线来表达。至于逻辑蓝图，我们使用 Rational Rose 来捕获并管理对象场景。

图 6-34 是关于场景的例子，显示了小型 PABX 的场景片段。相应的脚本如下。

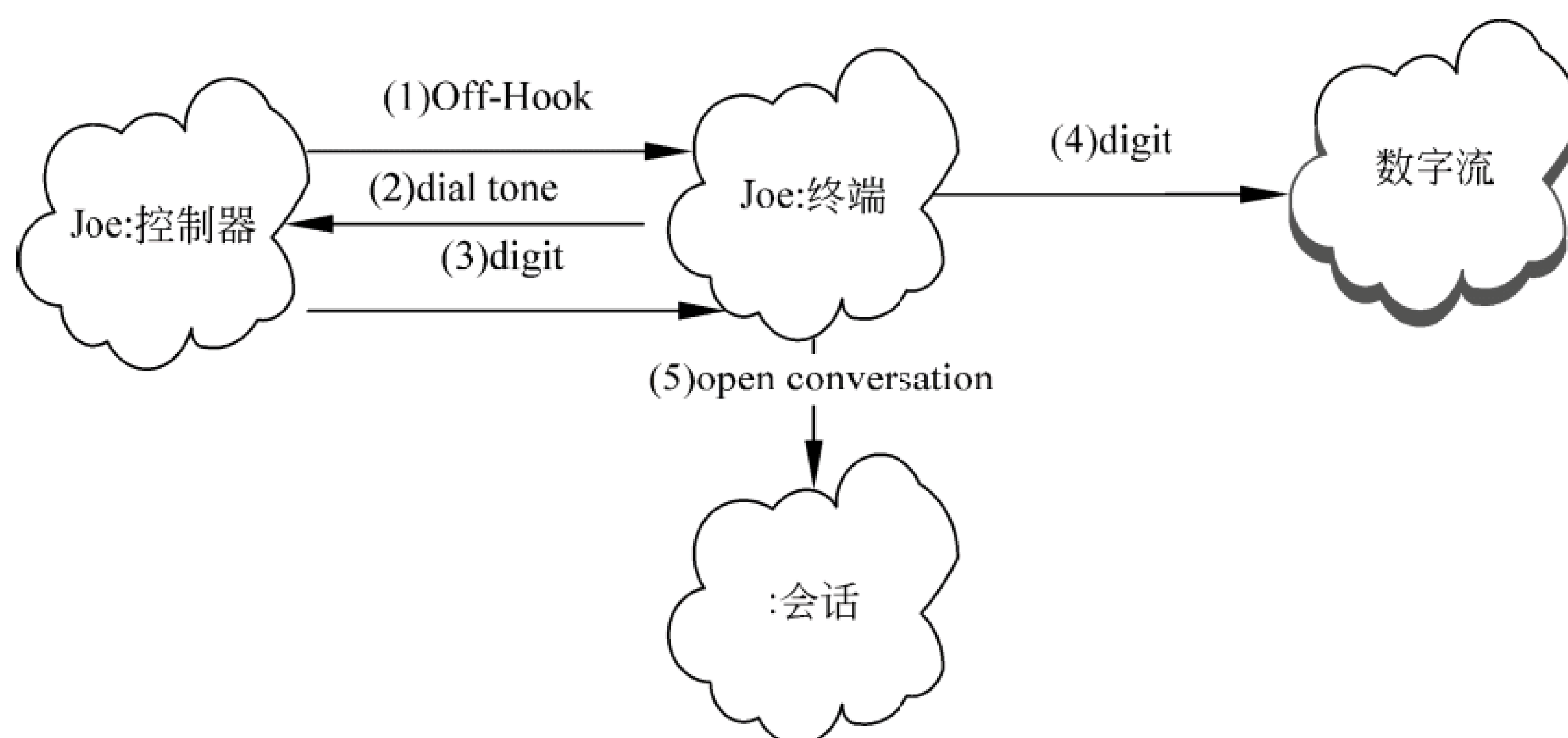


图 6-34　本地呼叫的初期场景——阶段选择

（1）Joe 的电话控制器检测和校验摘机状态的变换，并发送消息唤醒相应的终端对象。

（2）终端分配一些资源，并要求控制器发出拨号音。

（3）控制器接受拨号并传递给终端。

（4）终端使用拨号方案来分析数字流。

（5）有效的数字序列被键入，终端开始会话。

6.4.7　迭代过程

在进行文档化时，提倡一种更具有迭代性质的方法——架构先被原形化、测试、估量、分析，然后在一系列的迭代过程中被细化。该方法除了减少与架构相关的风险之外，对于项目而言还有其他优点：团队合作、培训，加深对架构的理解，深入程序和工具等（此处提及的是演进的原形，逐渐发展成为系统，而不是一次性的试验性的原形）。这种迭代方法还能够使需求被细化、成熟化并能够被更好地理解。

场景驱动（scenario-driven）的方法

系统大多数关键的功能以场景（或 use cases）的形式被捕获。关键意味着：最重要的功能，系统存在的理由，或使用频率最高的功能，或体现了必须减轻的一些重要的技术风险。

1）开始阶段

（1）基于风险和重要性为某次迭代选择一些场景。场景可能被归纳为对若干用户需

求的抽象。

（2）形成“稻草人式的架构”。然后对场景进行“描述”，以识别主要的抽象（类、机制、过程、子系统），如 Rubin 与 Goldberg6 所指出的——分解成为序列对（对象、操作）。

（3）所发现的架构元素被分布到 4 个蓝图中，即逻辑蓝图、进程蓝图、开发蓝图和物理蓝图。

（4）然后实施、测试、度量该架构，这项分析可能检测到一些缺点或潜在的增强要求。

（5）捕获经验教训。

2）循环阶段

（1）下一个迭代过程开始进行。

（2）重新评估风险。

（3）扩展考虑的场景选择板。

（4）选择能减轻风险或提高结构覆盖的额外的少量场景。

（5）然后试着在原先的架构中描述这些场景。

（6）发现额外的架构元素，或有时还需要找出适应这些场景所需的重要架构变更。

（7）更新 4 个主要视图：逻辑视图、进程视图、开发视图和物理视图。

（8）根据变更修订现有的场景。

（9）升级实现工具（架构原型）来支持新的、扩展了的场景集合。

（10）测试。如果可能的话，在实际的目标环境和负载下进行测试。

（11）然后评审这 5 个视图来检测简洁性、可重用性和通用性的潜在问题。

（12）更新设计准则和基本原理。

（13）捕获经验教训。

（14）终止循环。

为了实际的系统，初始的架构原型需要进行演进。较好的情况是在经过两次或三次迭代之后，结构变得稳定：主要的抽象都已被找到。子系统和过程都已经完成，以及所有的接口都已经实现。接下来则是软件设计的范畴，这个阶段可能也会用到相似的方法和过程。

这些迭代过程的持续时间参差不齐，原因在于：所实施项目的规模，参与项目人员的数量、他们对本领域和方法的熟悉程度，以及该系统和开发组织的熟悉程度等。因而较小的项目迭代过程可能持续 2～3 周，而大型的项目可能为 6～9 个月。

第 7 章　设 计 模 式

近年来，在面向对象领域中的一个重要突破就是提出了设计模式的概念。软件的设计模式是人们在长期的开发实践中良好经验的结晶，它提供了一个简单、统一的描述方法，使人们可以复用这些软件设计方法、过程管理经验。由于设计模式在表达上既经济又清楚，从而越来越受到重视。本章将介绍软件设计模式的概念、组成要素和分类，并介绍了 façade、Adapter、Abstract Factory 等常用设计模式。

7.1　设计模式概述

在任何设计活动中都存在着某些重复遇到的典型问题，不同开发人员对这些问题设计出不同的解决方案，随着设计经验在实践者之间日益广泛地被利用，描述这些共同问题和解决这些问题的方案就形成了所谓的模式。

7.1.1　设计模式的历史

模式概念是建筑师 Christopher Alexander 提出的，他提出可以把现实中一些已经实现的较好的建筑和房屋的设计经验作为模式，在以后的设计中直接加以运用。他还定义了一种“模式语言”来描述建筑和城市中的成功的架构。

Christopher Alexander 将模式分为几个部分：首先是特定的情景（Context），指模式在何种状况下发生作用；其二是动机（System of Force），指问题或预期的目标；其三是解决方案（Solution），指平衡各动机或解决所阐述问题的一个构造或配置。他提出模式是表示特定的情景、动机、解决方案三个方面关系的一个规则，每个模式描述了一个在某种特定情景下不断重复发生的问题，以及该问题解决方案的核心所在。模式既是一个事物又是一个过程，不仅描述该事物本身，而且提出了通过怎样的过程来产生该事物。设计模式的核心是问题描述和解决方案，问题描述说明模式的最佳使用场合及它将如何解决问题，解决方案是用一组类和对象及其结构和动态协作来描述的。

20 世纪 80 年代中期由 Ward Cunningham 和 Kent Beck 将其思想引入到软件领域。1995 年，E. Gamma，R.Helm，R. Johnson 和 J. Vlissides4 人合著了 *Design Patterns; Elements of Object-0riented Software*，这是软件设计模式领域中的一本经典书籍，从此设计模式成为软件工程领域内的一个重要研究领域，这四人也因此被称为 Gang of Four（GoF），成为设计模式中的大师级人物。

7.1.2 为什么要使用设计模式

面向对象设计时需要考虑许多因素，例如封装性、粒度大小、依赖关系、灵活性和可重用性等。如何确定系统中类及类之间的关系？如何保证在系统内部的一个类始终只有一个实例被创建？如何动态地将追加的功能增加到一个对象？哪些是设计时要努力达到的目标？这些都是软件设计中不容易掌握的问题。要真正掌握软件设计，必须研究其他软件设计大师的设计，这些设计中包含了许多设计模式。软件模式的应用对软件开发产生了重大的作用，主要表现在以下几个方面。

1．简化并加快设计

开发人员面对的问题来自不同的层次。在最底层，涉及的是单个类的接口或实现的细节问题；在最高层，涉及的是系统的整体架构的创建问题。设计模式关注的是中间层，在这一层必须保证局部化的特定的设计性质。设计模式使得软件开发人员无须从底层做起，开发人员可以重用成功的设计，可节省开发时间，同时有助于提高软件质量。

2．方便开发人员之间的通信

利用设计模式可以更准确地描述问题及问题的解决方案，使解决方案具有一致性；也有利于开发人员可以在更高层次上思考问题和讨论方案。例如，如果所有人都理解Factory 设计模式的意思，则开发人员可以用“建议采用 Factory 设计模式来解决这个问题”这样的话来表达。

3．降低风险

由于设计模式经过很多人的使用，已被证明是有效的解决方法，所以采用设计模式可以降低失败的可能性，也有利于在复杂的系统中产生简洁、精巧的设计。

4．有助于转到面向对象技术

新技术要在一个开发机构中得到应用，一般要经历两个阶段，即技术获取阶段和技术迁移阶段。技术获取阶段较容易，但在技术迁移阶段，由于开发人员对新技术往往会有抵触或排斥心理，对新技术可能带来的效果持怀疑态度，同时由于对新技术还是一知半解，所以要在一个开发机构中进行技术迁移并不是一件容易的事。设计模式一般都是基于面向对象技术而提出的，也可应用于接口定义良好的结构化方法中。另外，设计模式是可重用的设计经验的总结，已在实际的系统中多次得到成功应用，因此通过对设计模式的研究，能够深入理解良好设计的最基本的性质，从而有助于说服开发人员采用新技术。

成熟的软件设计模式具有以下特性。

（1）巧妙：设计模式是一些优雅的解决方案，是在大量实践经验的基础上提炼出来的。

（2）通用：设计模式通常不依赖于某个特定的系统类型、程序设计语言或应用领域，它们是通用的。

（3）得到很好的证明：设计模式在实际系统和面向对象系统中得到广泛应用，它们并不仅仅停留在理论上。

（4）简单：设计模式通常都非常简单，只涉及很少的一些类。为了构建更多更复杂的解决方案，可以把不同的设计模式与应用代码结合或混合使用。

（5）可重用：设计模式的建档方式使它们非常易于使用，因而可方便用于任何适宜的系统。

（6）面向对象：设计模式是用最基本的面向对象机制如类、对象、多态等构造的。许多模式特别强调了某些面向对象设计擅长的领域，例如，区分接口和实现、降低各部分之间的依赖性、隔离硬件和软件等。

7.1.3　设计模式的组成元素

模式是一个高度抽象的概念。设计模式的基本组成元素如下。

1. 模式名

模式必须具有一个有意义的名称，这样就可以用一个词或短语来指代该模式，以及它所描述的知识和结构。模式名称简洁地描述了模式的本质。模式名可以帮助我们思考，便于我们与其他人交流设计思想及设计结果，找到恰当的模式名也是设计模式编目工作的难点之一。

2. 问题或意图

陈述问题并描述它的意图，以及它在特定的情景和动机下要达到的目标，它解释了设计问题和问题存在的前因后果，它可能描述了特定的设计问题，如怎样用对象表示算法等，也可能描述了导致不灵活设计的类或对象结构。有时候，问题部分会包括使用模式必须满足的一系列先决条件。通常情况下这些动机和目标是相互矛盾、相互影响的。

3. 情景

情景是问题及其解决方案产生时的前提条件。情景告诉我们该模式的适用性，可以将情景视为应用该模式之前的系统初始配置。

4. 动机

它描述相关的动机和约束，它们之间或与期望达到的目标之间的相互作用（或冲突），通常需要对各期望的目标进行优先级排序。动机阐明了问题的复杂性，定义了在相互冲突时所采取的各种权衡手段。一个好的模式应尽可能将所有产生影响的动机考虑在内。

5. 解决方案

解决方案是描述一些静态的关系和动态的规则，用以描述如何得到所需的结果。通常是给出一组指令来说明如何构造所需的工作制品。该说明可包括图表、文字，用以标示模式的结构、参与者及其之间的协作，从而表明问题是如何解决的。因为模式就像一个模板，可应用于多种不同的场合，所以解决方案并不描述一个特定而具体的设计或实现，而是提供设计问题的抽象描述和怎样用一个具有一般意义的元素组合（类或对象组

合）来解决这个问题。

6．示例

示例指一个或多个该模式的应用例子，示例说明了模式在怎样的初始情景下如何发生作用，如何改变情景而导致结果情景的出现。示例帮助读者理解模式的具体使用方法。

7．结果情景

结果情景指在应用该模式后系统的状态或配置，包括模式发生作用后带来的后果，以及在新的情景下产生的问题、可应用的模式等。它阐述了模式的后续状态和副作用。通常通过对结果情景的描述，使该模式与其他模式联系起来（该模式的结果情景成为其他模式的初始情景）。

8．基本原理

基本原理指对该模式中的解决步骤或采用的规则的解释、证明，解释该模式如何、为何能解决当前问题，它采用的方法为何能得到与期望相一致的结果。

9．相关模式

该模式与其他模式的关系，包括静态的和动态的。例如，该模式的前导模式（前导模式应用后产生的结果情景与该模式的初始情景一致）、后续模式（该模式应用后产生的结果情景与后续模式的初始情景一致）、替代模式（使用该模式的替代模式产生同样的效果）等。

10．已知应用

阐述该模式在已有应用系统中的实际应用情况，有助于验证该模式的有效性。尽管我们描述设计决策时，并不总提到模式效果，但它们对于评价设计选择和理解使用模式的代价及好处具有重要意义。模式效果大多关注对时间和空间的衡量，它们也表述了语言和实现问题。因为复用是面向对象设计的要素之一，所以模式效果包括它对系统的灵活性、扩充性或可移植性的影响，显式地列出这些效果对理解和评价这些模式很有帮助。

通常好的模式前面都有一个摘要，提供简短的总结和概述，为模式描绘出一个清晰的图画，提供有关该模式能够解决问题的快速信息。有时这种描述称为模式的缩略概要，或一个缩略图。模式应该说明它的目标读者，以及对读者有哪些知识要求。

7.1.4 设计模式的分类

软件模式主要可分为设计模式、分析模式、组织和过程模式等，每一类又可细分为若干个子类。在此着重介绍设计模式，目前它的使用最为广泛。设计模式主要用于得到简洁灵活的系统设计，GoF 的书中共有 23 个设计模式，这些模式可以按两个准则来分类：一是按设计模式的目的划分，可分为创建型、结构型和行为型三种模式；二是按设计模式的范围划分，即根据设计模式是作用于类还是作用于对象来划分，可以把设计模式分为类设计模式和对象设计模式。

1．创建型模式

该类型模式是对对象实例化过程的抽象，它通过采用抽象类所定义的接口，封装了

系统中对象如何创建、组合等信息。

2．结构型模式

该类模式主要用于如何组合已有的类和对象以获得更大的结构，一般借鉴封装、代理、继承等概念将一个或多个类或对象进行组合、封装，以提供统一的外部视图或新的功能。

3．行为型模式

该类模式主要用于对象之间的职责及其提供的服务的分配，它不仅描述对象或类的模式，还描述它们之间的通信模式，特别是描述一组对等的对象怎样相互协作以完成其中任一对象都无法单独完成的任务。

7.2　设计模式实例

7.2.1　创建性模式

在系统中，创建性模式支持对象的创建。该模式允许在系统中创建对象，而不需要在代码中标识特定类的类型，这样用户就不需要编写大量、复杂的代码来初始化对象。它是通过该类的子类来创建对象的。但是，这可能会限制在系统内创建对象的类型或数目。本节将介绍如下的创建性模式：

- Abstract Factory（抽象工厂）。
- Builder（构建器）。
- Factory Method（工厂方法）。
- Prototype（原型）。
- Singleton（单独）。

1．Abstract Factory 模式

在不指定具体类的情况下，这种模式为创建一系列相关或相互依赖的对象提供了一个接口。根据给定的相关抽象类，Abstract Factory 模式提供了从一个相匹配的具体子类集创建这些抽象类的实例的方法，如图 7-1 所示。

Abstract Factory 模式提供了一个可以确定合适的具体类的抽象类，这个抽象类可以用来创建实现标准接口的具体产品的集合。客户端只与产品接口和 Abstract Factory 类进行交互。使用这种模式，客户端不用知道具体的构造类。Abstract Factory 模式类似于 Factory Method 模式，但是 Abstract Factory 模式可以创建一系列的相关对象。

其优点如下。

- 可以与具体类分开。
- 更容易在产品系列中进行转换。
- 提高了产品间的一致性。

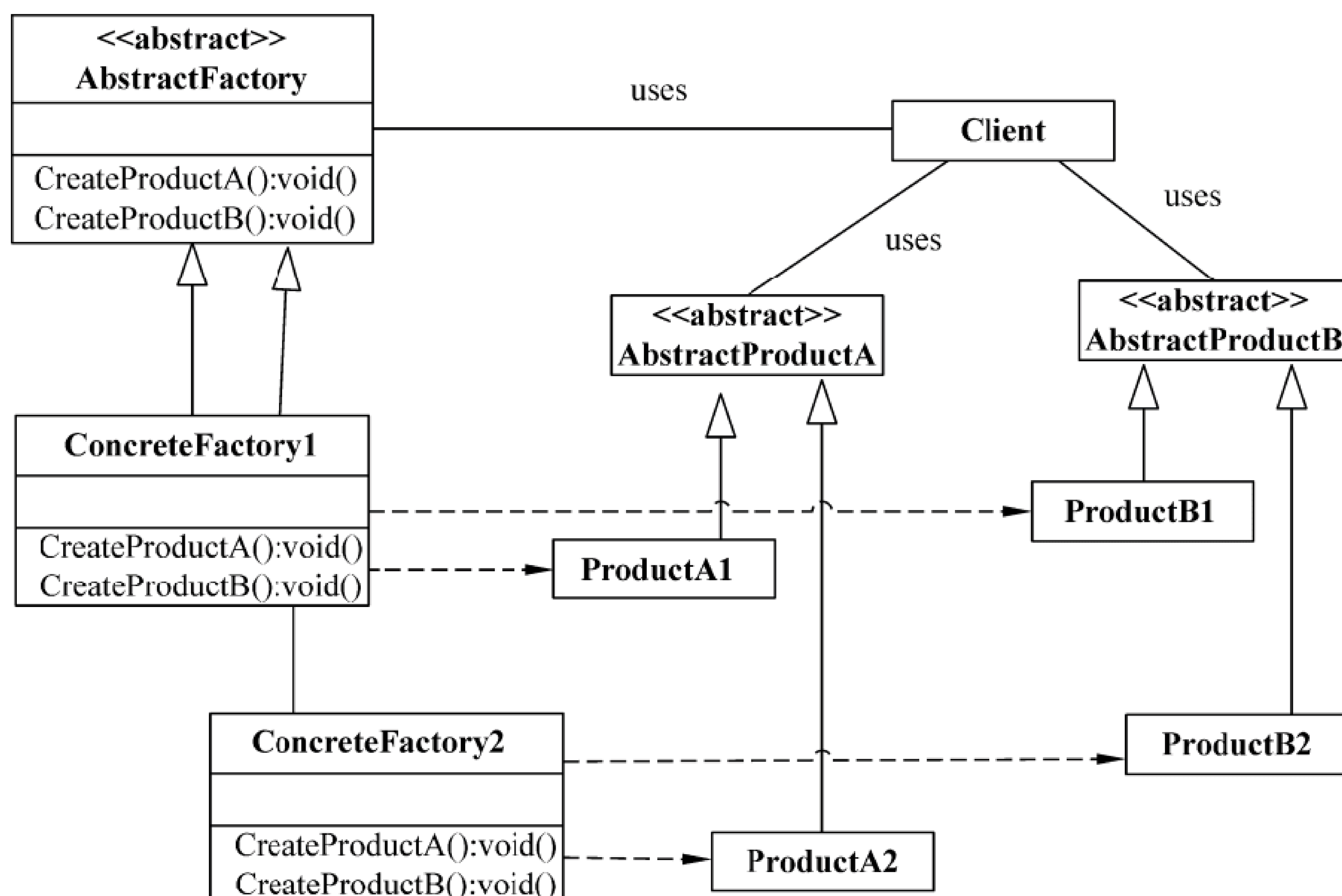


图 7-1　Abstract Factory 模式

在以下情况中，应该使用 Abstract Factory 模式：

- 系统独立于产品的创建、组成以及表示。
- 系统配置成具有多个产品的系列，例如 Microsoft Windows 或 Apple McIntosh 类。
- 相关产品对象系列是共同使用的，而且必须确保这一点。这是该模式的关键，否则可以使用 Factory Method 模式。
- 你希望提供产品的类库，只开放其接口，而不是其实现。

2．Builder 模式

Builder 模式将复杂对象的构建与其表示相分离，这样相同的构造过程可以创建不同的对象。通过只指定对象的类型和内容，Builder 模式允许客户端对象构建一个复杂对象。客户端可以不受该对象构造的细节的影响。这样通过定义一个能够构建其他类实例的类，就可以简化复杂对象的创建过程。Builder 模式生产一个主要产品，而该产品中可能有多个类，但是通常只有一个主类。图 7-2 所示的就是 Builder 模式，当使用该模式时，可以一次就创建所有的复杂对象。而其他模式一次就只能创建一个对象。

其优点如下。

- 可以对产品的内部表示进行改变。
- 将构造代码与表示代码相分离。

在以下情况中，应该使用 Builder 模式：

- 创建复杂对象的算法独立于组成对象的部分以及这些部分的集合方式。
- 构造过程必须允许已构建对象有不同表示。

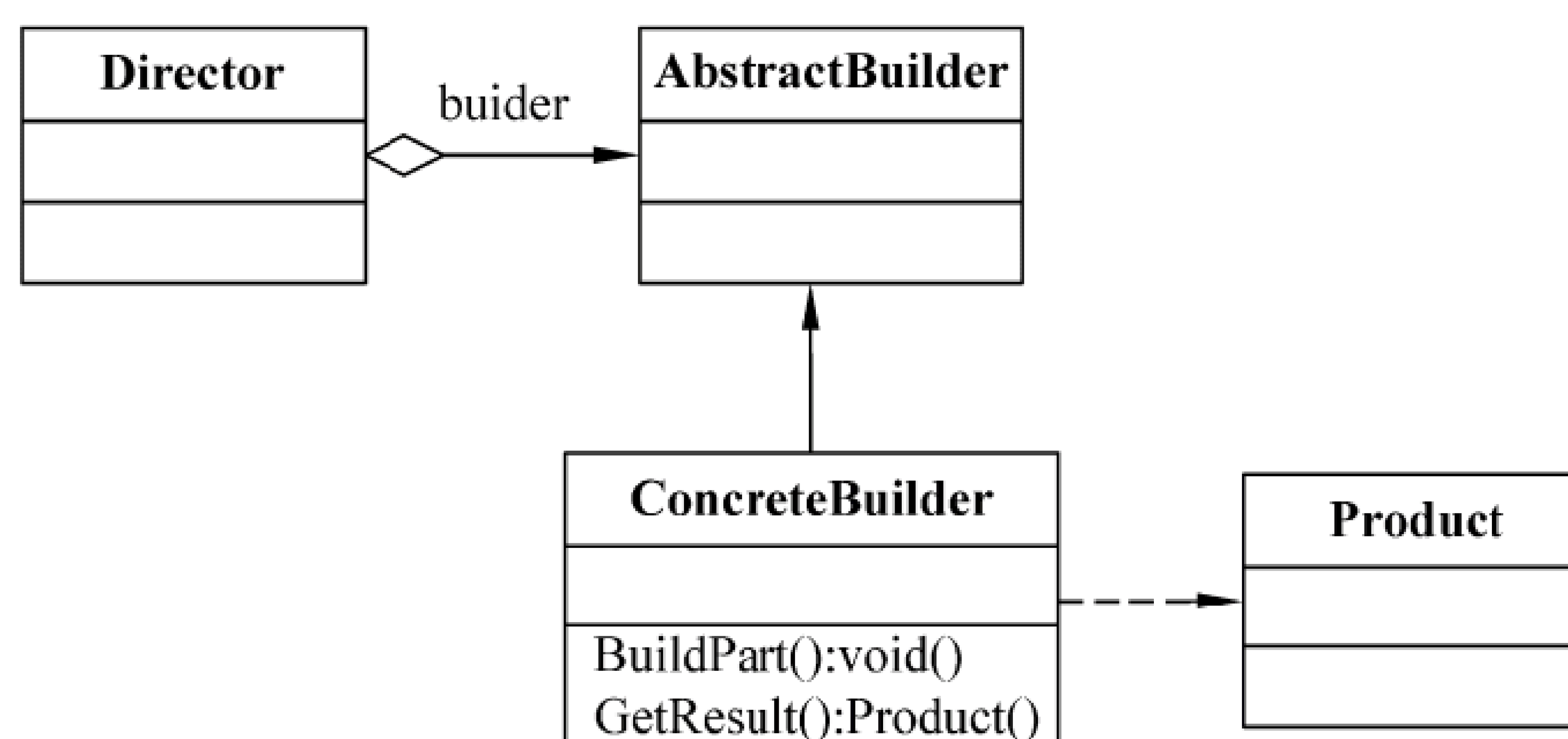


图 7-2　Builder 模式

3. Factory Method 模式

Factory Method 模式定义了创建对象的接口，它允许子类决定实例化哪个类。它允许类将实例化工作交给其子类，这对于在特定目的下构建单个对象是非常有帮助的，而且它不需要请求者知道要被实例化的特定类，这就可以在不修改代码的情况下引入新类，因为新类只实现了接口，这样它就可以被客户端使用。可以创建一个新的 Factory 类来创建新类，而由这个 Factory 类来实现 Factory 接口。图 7-3 所示的是 Factory Method 模式。

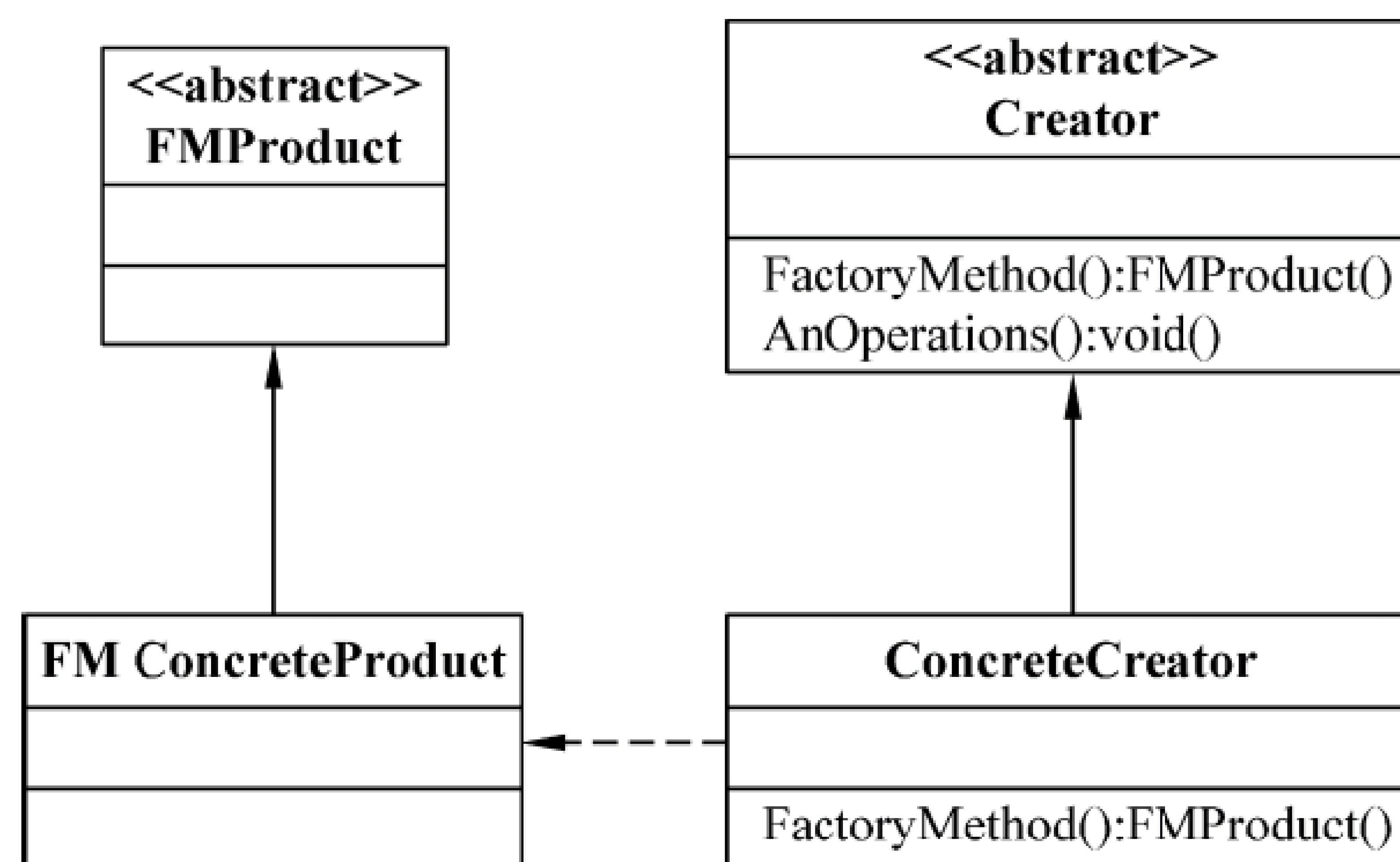


图 7-3　Factory method 模式

其优点如下。

- 没有了将应用程序类绑定到代码中的要求，代码只处理接口，因此可以使用任何实现了接口的类。
- 允许子类提供对象的扩展版本，因为在类中创建对象比直接在客户端创建要更加灵活。

在以下情况中，应该使用 Factory Method 模式：

- 类不能预料它必须创建的对象的类。
- 类希望其子类指定它要创建的对象。

- 类将责任转给某个帮助子类，而用户希望定位那个被授权的帮助子类。

4．Prototype 模式

Prototype 模式允许对象在不了解要创建对象的确切类以及如何创建等细节的情况下创建自定义对象。使用 Prototype 实例，便指定了要创建的对象类型，而通过复制这个 Prototype，就可以创建新的对象。Prototype 模式是通过先给出一个对象的 Prototype 对象，然后再初始化对象的创建。创建初始化后的对象再通过 Prototype 对象对其自身进行复制来创建其他对象。Prototype 模式使得动态创建对象更加简单，只要将对象类型定义成能够复制自身就可以实现。图 7-4 所示的是 Prototype 模式。

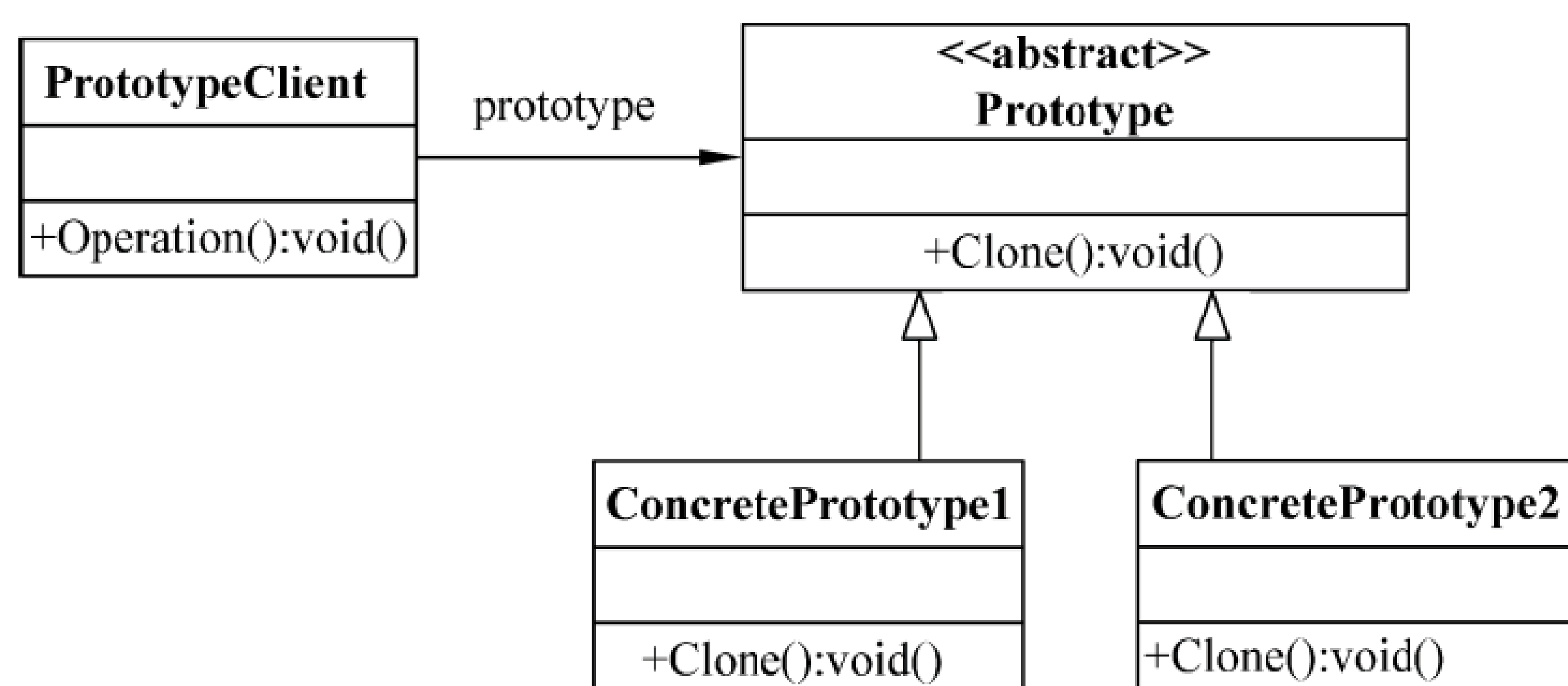


图 7-4　Prototype 模式

其优点如下。

- 可以在运行时添加或删除产品。
- 通过改变值指定新对象。
- 通过改变结构指定新对象。
- 减少子类的生成和使用。
- 可以用类动态地配置应用程序。

在以下情况中，应该使用 Prototype 模式：

- 在运行时，指定需要例化的类，例如动态载入。
- 避免构建与产品的类层次结构相似的工厂类层次结构。
- 当类的实例是仅有的一些不同状态组合之一的时候。

5．Singleton 模式

Singleton 模式确保一个类只有一个实例，并且提供了对该类的全局访问入口，它可以确保使用这个类实例的所有的对象使用相同的实例。图 7-5 所示的是 Singleton 模式。

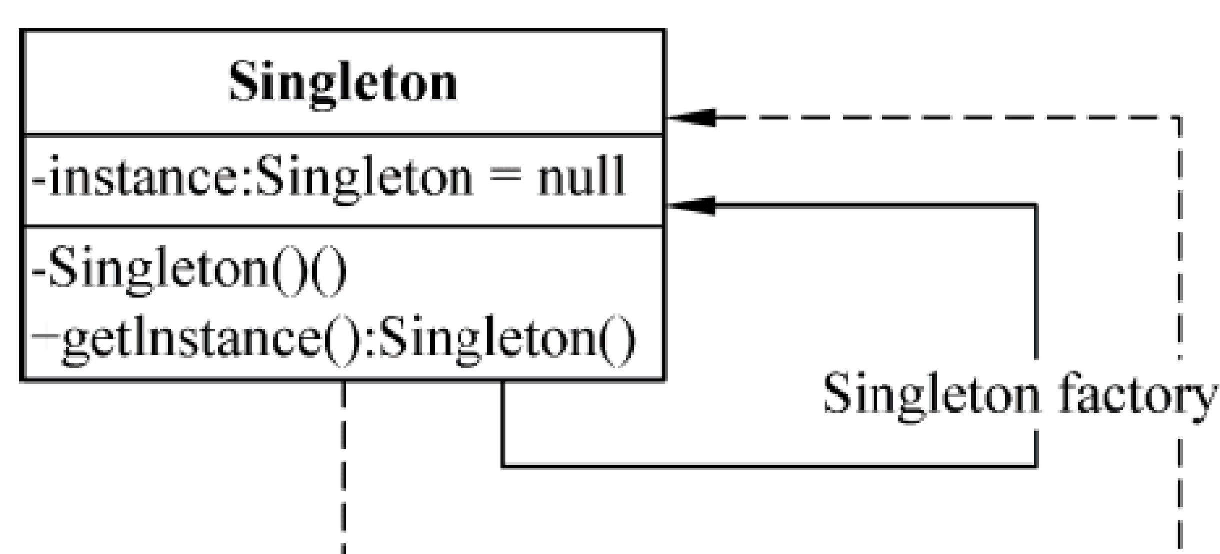


图 7-5　Singleton 模式

其优点如下。

- 对单个实例的受控制访问。

- 名称空间的减少。
- 允许改进操作和表示。
- 允许可变数目的实例。
- 比类操作更灵活。

在以下情况中，应该使用 Singleton 模式：如只有一个类实例。

7.2.2　结构性模式

结构性模式控制了应用程序较大部分之间的关系。它将以不同的方式影响应用程序，例如 Adapter 模式允许两个不兼容的系统进行通信，而 Facade 模式允许在不删除系统中所有可用选项的情况下为用户提供一个简化的界面。结构性模式允许在不重写代码或自定义代码的情况下创建系统。这可以使系统具有增强的重复使用性和应用性能。本节将介绍如下的结构性模式：

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

1．Adapter 模式

Adapter 模式可以充当两个类之间的媒介，它可以转换一个类的接口，这样就可以被另外一个类使用，这使得具有不兼容接口的类能够协同使用。Adapter 模式实现为客户端所知的接口，并且为客户端提供对不为其所知的类实例的访问。Adapter 对象可以在不知道实现该接口的类的情况下提供该接口的功能。图 7-6 所示的就是 Adapter 模式。

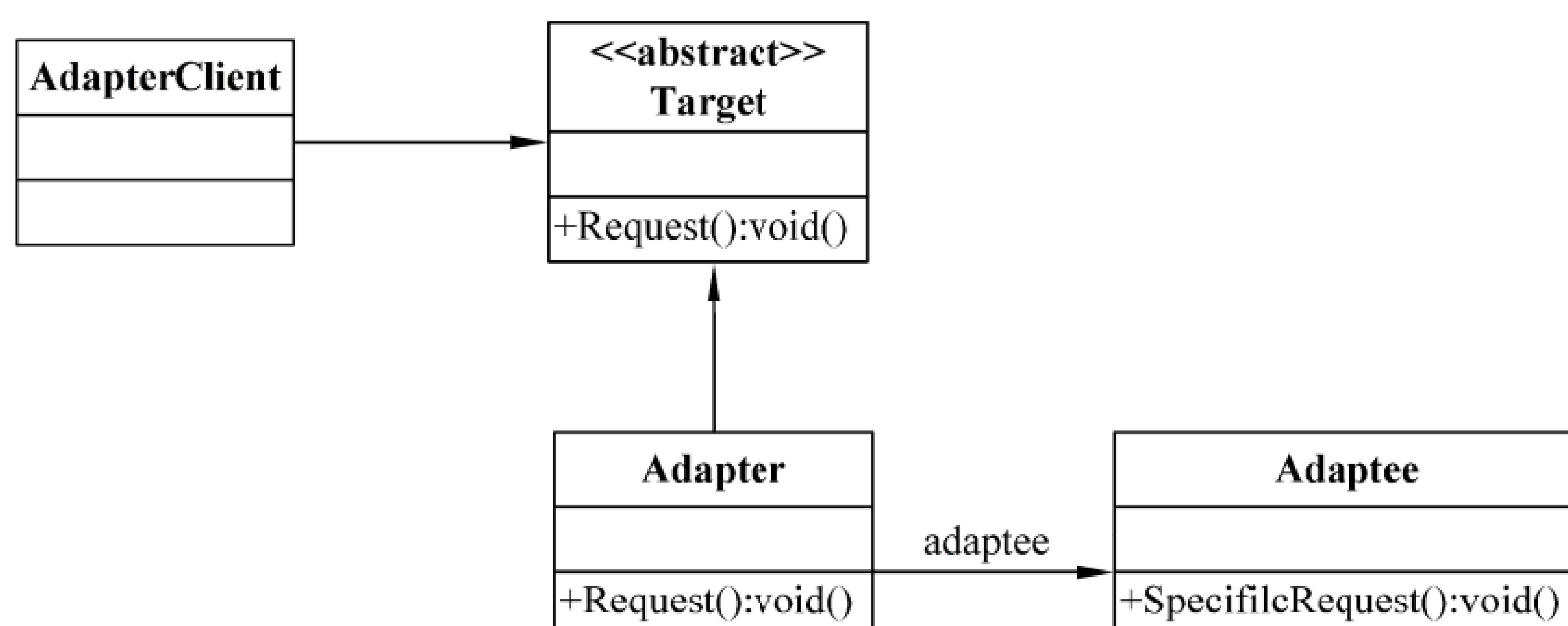


图 7-6　Adapter 模式

其优点如下。

- 允许两个或多个不兼容的对象进行交互和通信。

- 提高已有功能的重复使用性。

在以下情况中，应该使用 Adapter 模式：

- 要使用已有类，而该类接口与所需的接口并不匹配。
- 要创建可重用的类，该类可以与不相关或未知类进行协作，也就是说，类之间并不需要兼容接口。
- 要在一个不同于已知对象接口的接口环境中使用对象。
- 必须要进行多个源之间的接口转换的时候。

2．Bridge 模式

Bridge 模式可以将一个复杂的组件分成两个独立的但又相关的继承层次结构：功能性的抽象和内部实现。改变组件的这两个层次结构很简单，以至于它们可以相互独立地变化。当具有抽象的层次结构和相应的实现层次结构时，Bridge 模式是非常有用的。除了可以将抽象和实现组合成许多不同的类，该模式还可以以动态组合的独立类的形式实现这些抽象和实现。图 7-7 所示的是 Bridge 模式。

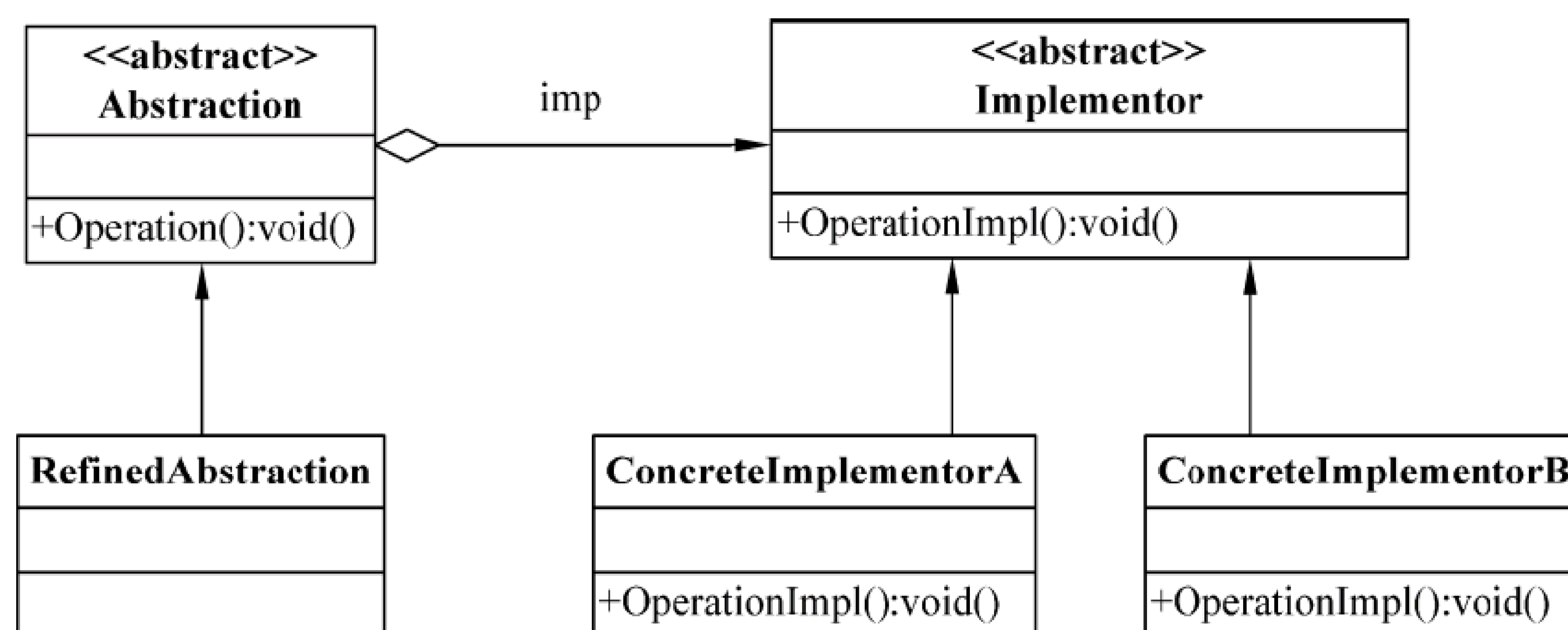


图 7-7 Bridge 模式

其优点如下。

- 可以将接口与实现相分离。
- 提高了可扩展性。
- 对客户端隐藏了实现的细节。

在以下情况中，应该使用 Bridge 模式：

- 想避免在抽象及其实现之间存在永久的绑定。
- 抽象及其实现可以使用子类进行扩展。
- 抽象的实现被改动应该对客户端没有影响；也就是说，你不用重新编译代码。

3．Composite 模式

Composite 模式允许创建树型层次结构来改变复杂性，同时允许结构中的每一个元素操作同一个接口。该模式将对象组合成树型结构来表示整个或部分的层次结构。这就意味着 Composite 模式允许客户端使用单个对象或多个同一对象的组合。图 7-8 所示的是 Composite 模式。

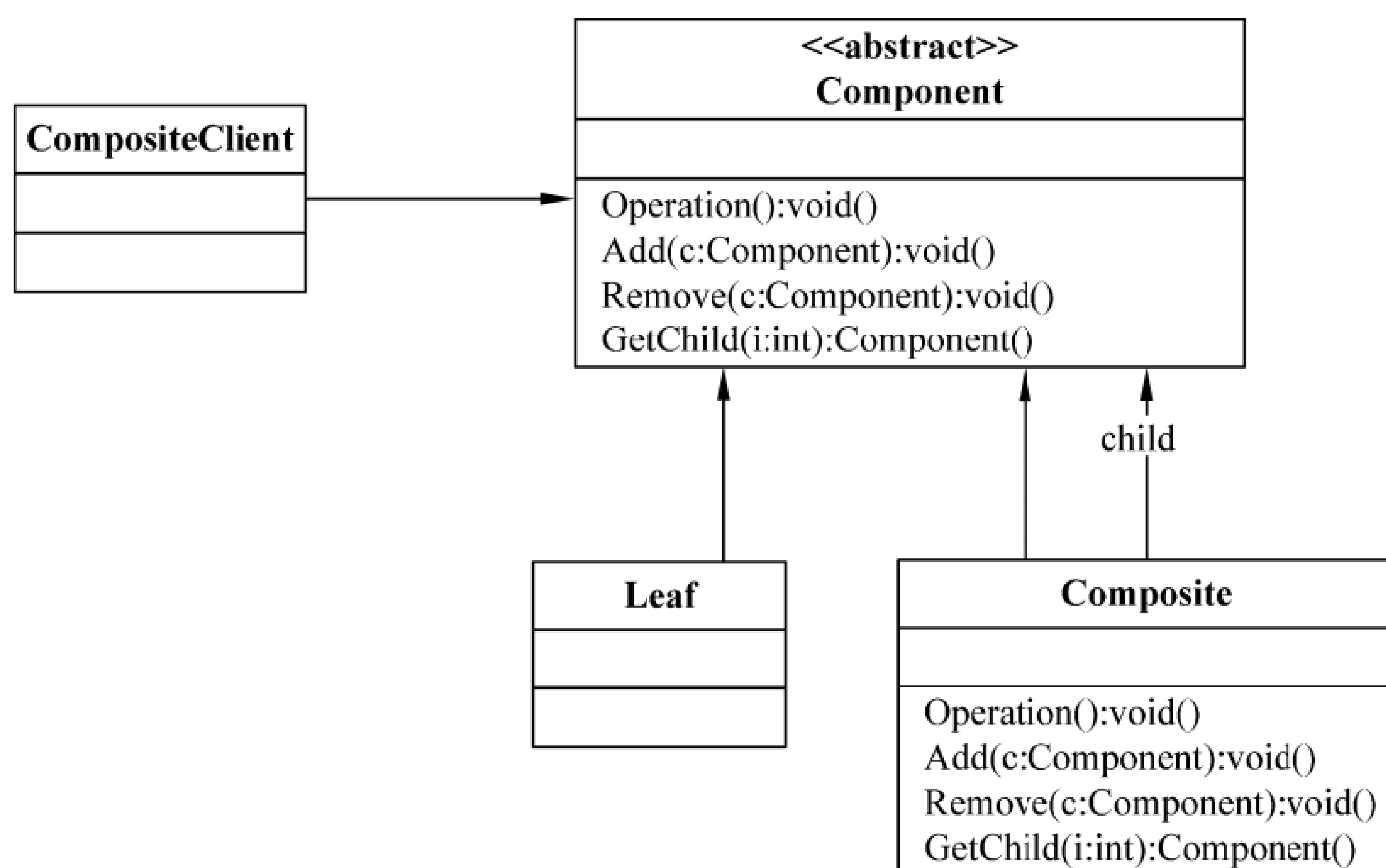


图 7-8　Composite 模式

其优点如下。

- 定义了由主要对象和复合对象组成的类层次结构。
- 使得添加新的组件类型更加简单。
- 提供了结构的灵活性和可管理的接口。

在以下情况中，应该使用 Composite 模式：

- 想要表示对象的整个或者部分的层次结构。
- 想要客户端能够忽略复合对象和单个对象之间的差异。
- 结构可以具有任何级别的复杂性，而且是动态的。

4．Decorator 模式

Decorator 模式可以在不修改对象外观和功能的情况下添加或者删除对象功能。它可以使用一种对客户端来说是透明的方法来修改对象的功能，也就是使用初始类的子类实例对初始对象进行授权。Decorator 模式还为对象动态地添加了额外的责任，这样就在不使用静态继承的情况下，为修改对象功能提供了灵活的选择。图 7-9 所示的是 Decorator 模式。

其优点如下。

- 比静态继承具有更大的灵活性。
- 避免了特征装载的类处于层次结构的过高级别。
- 简化了编码，因为用户编写的每一个类都针对功能的一个特定部分，而不用将所有的行为编码到对象中。
- 改进了对象的扩展性，因为用户可以通过编写新的类来作出改变。

在以下情况中，应该使用 Decorator 模式：

- 想要在单个对象中动态并且透明地添加责任，而这样并不会影响其他对象。

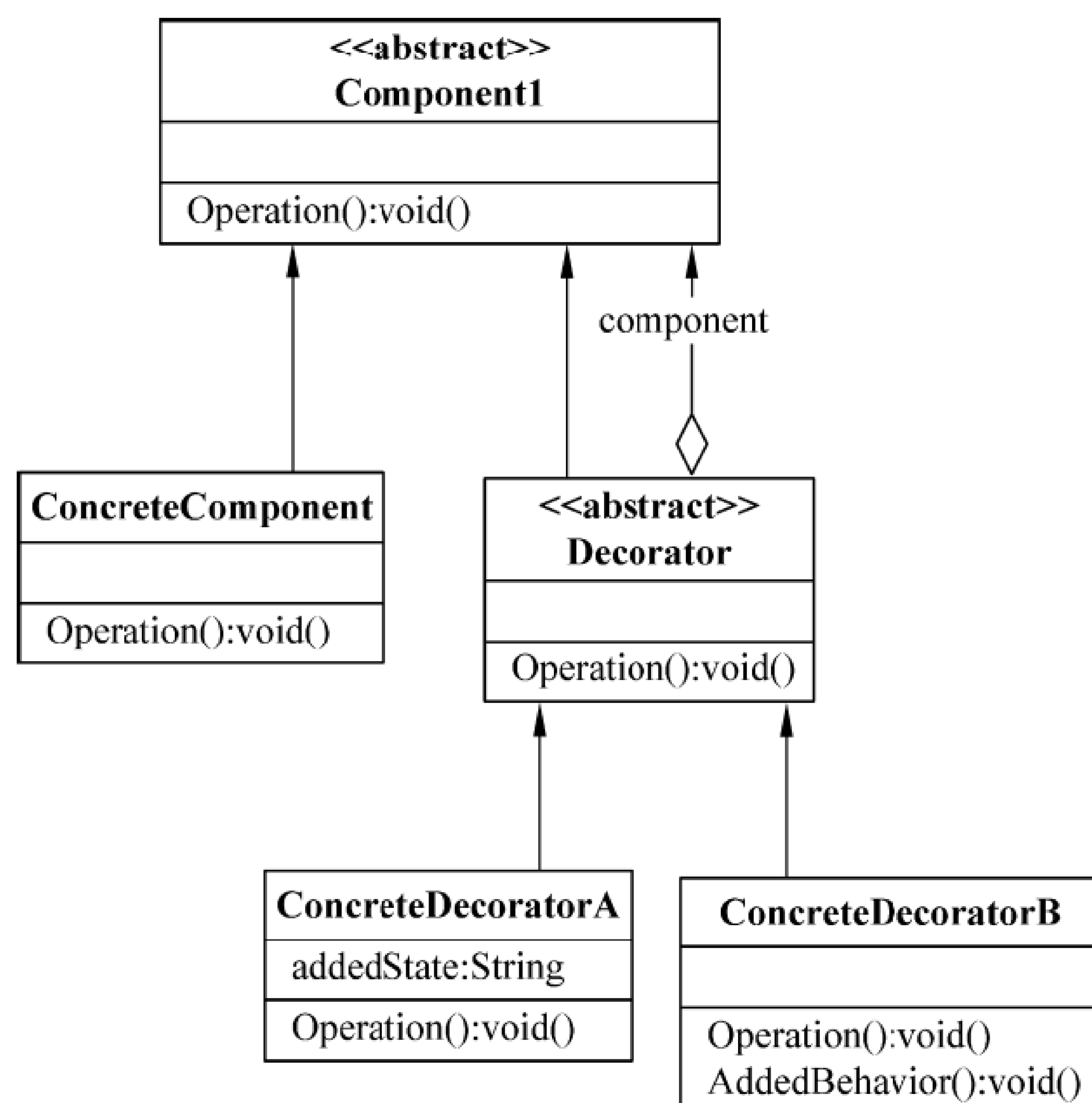


图 7-9　Decorator 模式

- 想要在以后可能要修改的对象中添加责任。
- 当无法通过静态子类化实现扩展时。

5. Facade 模式

Facade 模式为子系统中的一组接口提供了一个统一的接口。因为只有一个接口，该模式就定义了更容易使用子系统的高级接口。这个统一的接口允许对象使用该接口与子系统进行通信，从而实现对子系统的访问。图 7-10 所示的是 Facade 模式。

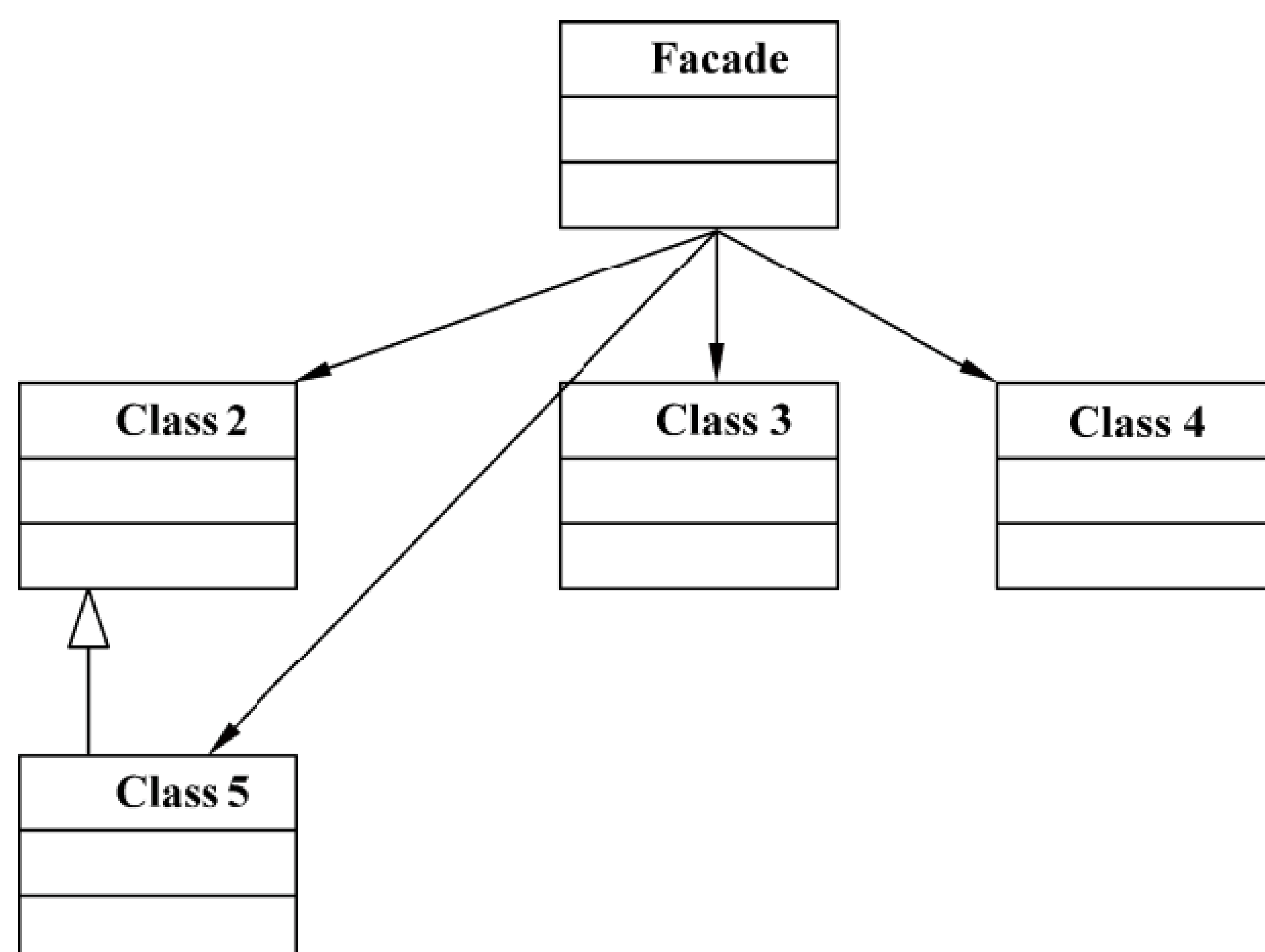


图 7-10　Facade 模式

其优点如下。

- 在不减少系统所提供的选项的情况下，为复杂系统提供了简单接口。
- 对客户端屏蔽了子系统组件。
- 提高了子系统与其客户端之间的弱耦合度。
- 如果每一个子系统使用自身的 Facade 模式而且系统的其他部分也使用 Facade 模式与子系统进行通信的话，就可以降低子系统之间的耦合度。
- 将客户端请求转换后发送给能够处理这些请求的子系统。

在以下情况中，应该使用 Facade 模式：

- 想要为复杂的子系统提供简单的接口。
- 在客户端和抽象的实现类中存在许多依赖关系。
- 想要对子系统进行分层。

6. Flyweight 模式

Flyweight 模式可以通过共享对象减少系统中低等级的、详细的对象数目。如果一个类实例包含用来互换使用的相同信息，Flyweight 模式允许程序通过共享一个接口来避免使用多个具有相同信息的实例所带来的开销。图 7-11 所示的是 Flyweight 模式。

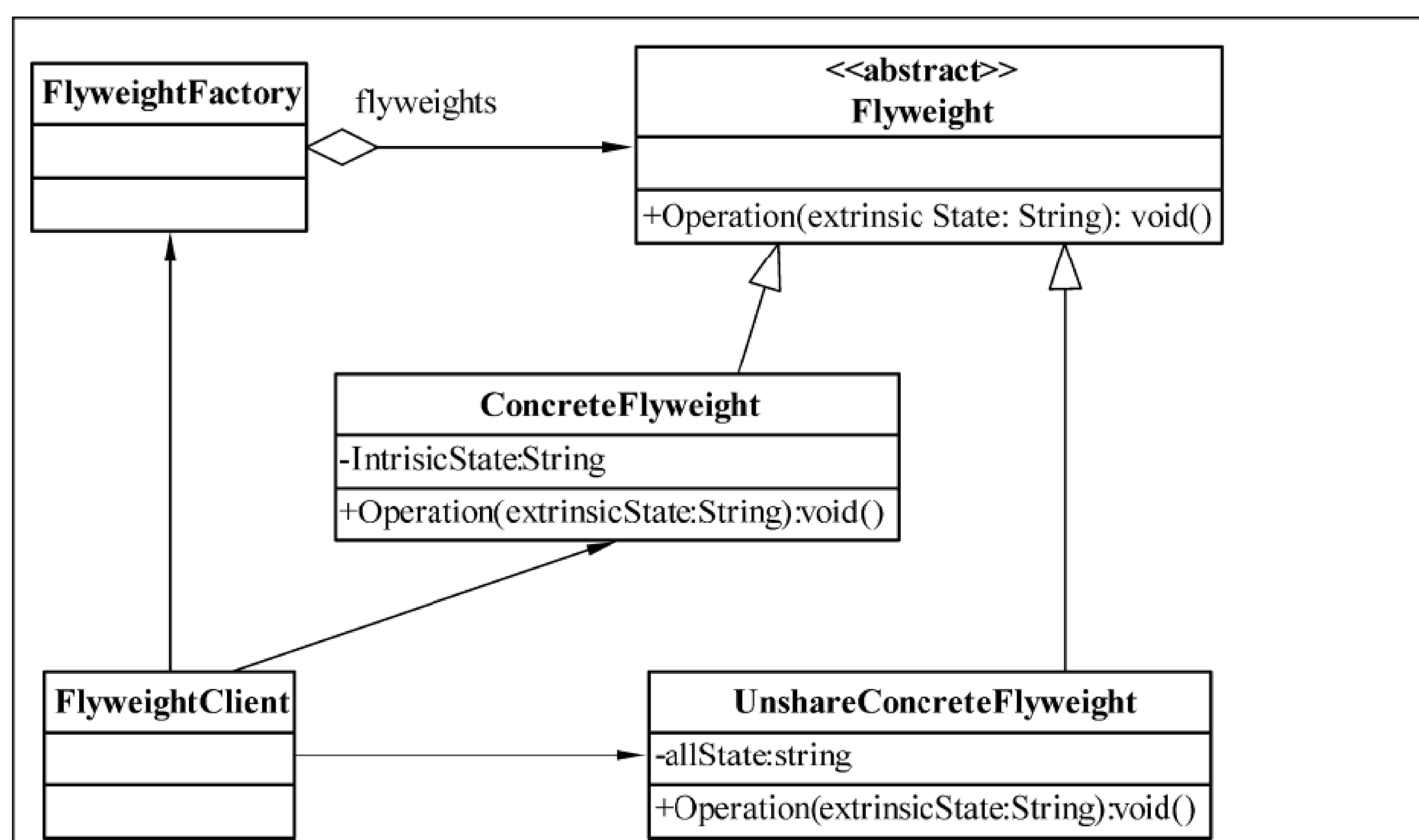


图 7-11　Flyweight 模式

其优点如下。

- 减少了要处理的对象数目。
- 如果对象能够持续，可以减少内存和存储设备。

在以下情况中，应该使用 Flyweight 模式：

- 应用程序使用大量的对象。
- 由于对象数目巨大，导致很高的存储开销。

- 应用程序不依赖于对象的身份。

7. Proxy 模式

Proxy 模式为控制对初始对象的访问提供了一个代理或者占位符对象。它的实现可以有多种类型，其中 Remote Proxy（远程代理）和 Virtual Proxy（虚拟代理）是最常见的。图 7-12 所示的是 Proxy 模式。

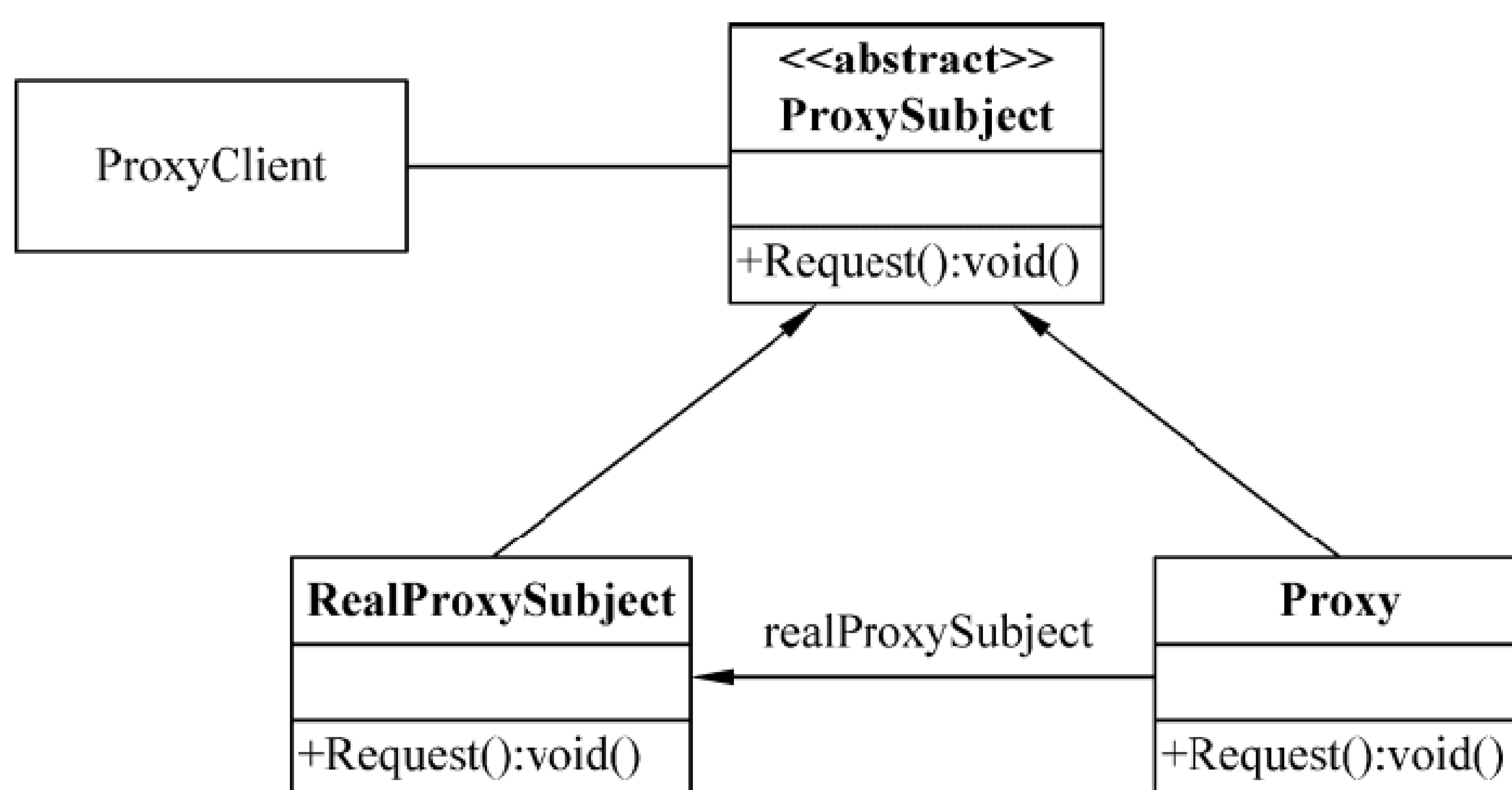


图 7-12 Proxy 模式

其优点如下。

- 远程代理可以隐藏对象位于不同的地址空间的事实。
- 虚拟代理可以执行优化操作，例如根据需要创建一个对象。

在以下情况中，应该使用 Proxy 模式，如需要比简单的指针更灵活、更全面的对象引用。

7.2.3 行为性模式

行为性模式可以影响一个系统的状态和行为流。通过优化状态和行为流转换和修改的方式，可以简化、优化并且提高应用程序的可维护性。本节将介绍如下的行为性模式：

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

1．Chain of Responsibility 模式

Chain of Responsibility 模式可以在系统中建立一个链，这样消息可以在首先接收到它的级别处被处理，或者可以定位到可以处理它的对象。图 7-13 所示的是 Chain of Responsibility 模式。

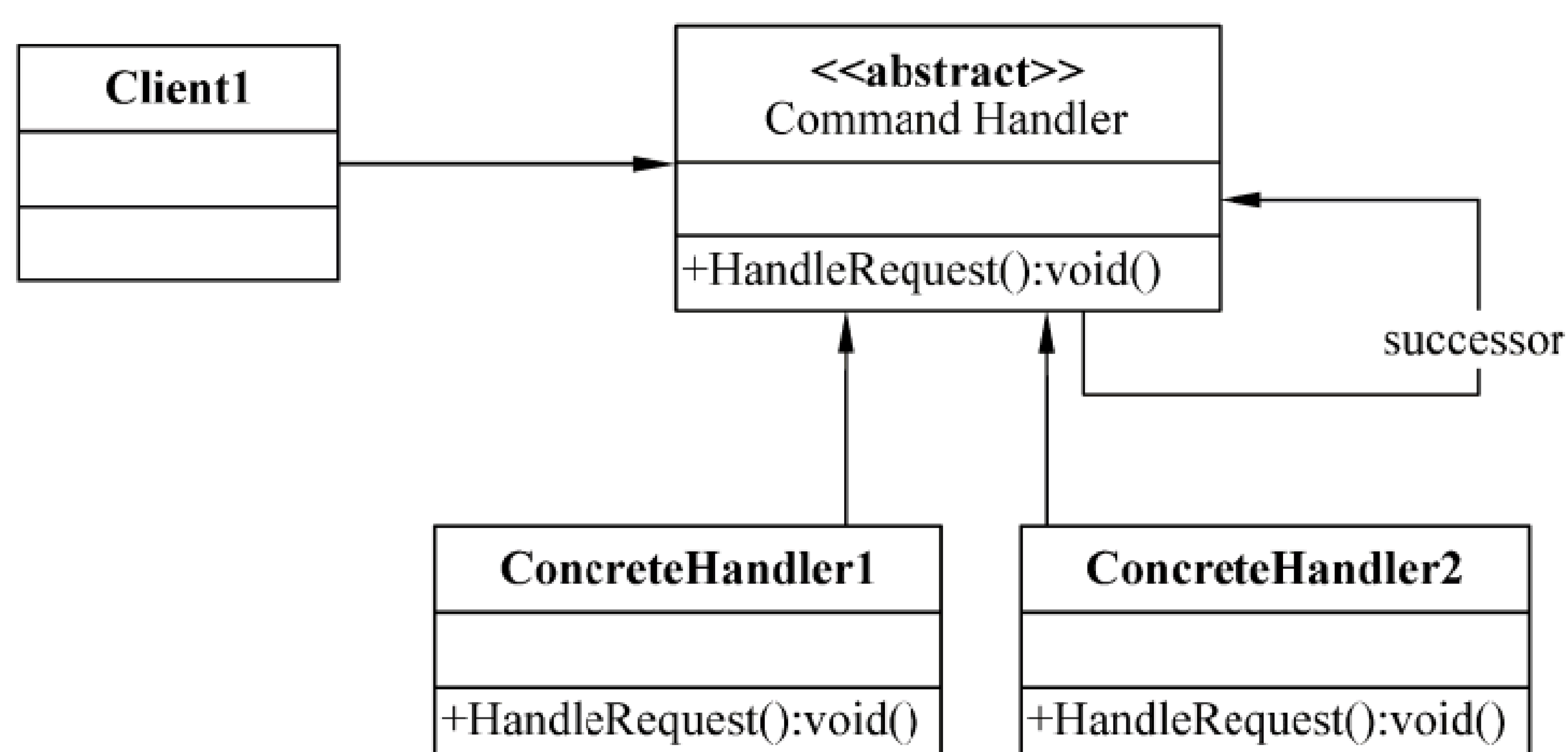


图 7-13　Chain of Responsibility 模式

其优点如下。

- 降低了耦合度。
- 增加向对象指定责任的灵活性。
- 由于在一个类中产生的事件可以被发送到组成中的其他类处理器上，类的集合可以作为一个整体。

在以下情况中，应该使用 Chain of Responsibility 模式：

- 多个对象可以处理一个请求，而其处理器却是未知的。
- 想要在不指定确切的请求接收对象的情况下，向几个对象中的一个发送请求。
- 可以动态地指定能够处理请求的对象集。

2．Command 模式

Command 模式在对象中封装了请求，这样就可以保存命令，将该命令传递给方法以及像任何其他对象一样返回该命令。图 7-14 所示的是 Command 模式。

其优点如下。

- 将调用操作的对象与知道如何完成该操作的对象相分离。
- 更容易添加新命令，因为不用修改已有类。

在以下情况中，应该使用 Command 模式：

- 想要通过要执行的动作来参数化对象。
- 要在不同的时间指定、排序以及执行请求。
- 必须支持 Undo、日志记录或事务。

3．Interpreter 模式

interpreter 模式可以解释定义其语法表示的语言，还提供了用表示来解释语言中的语

句的解释器。图 7-15 所示的是 Interpreter 模式。

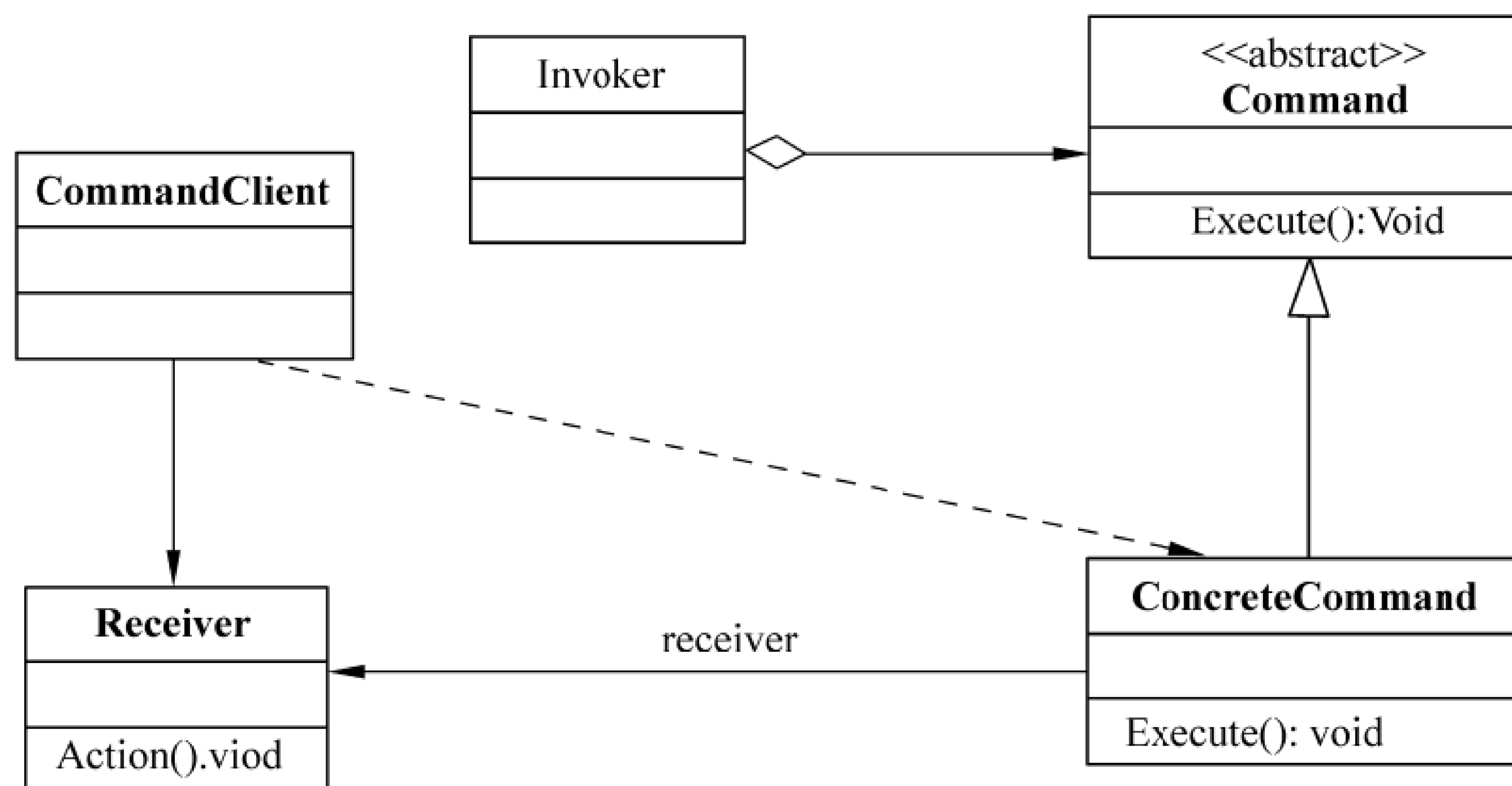


图 7-14 Command 模式

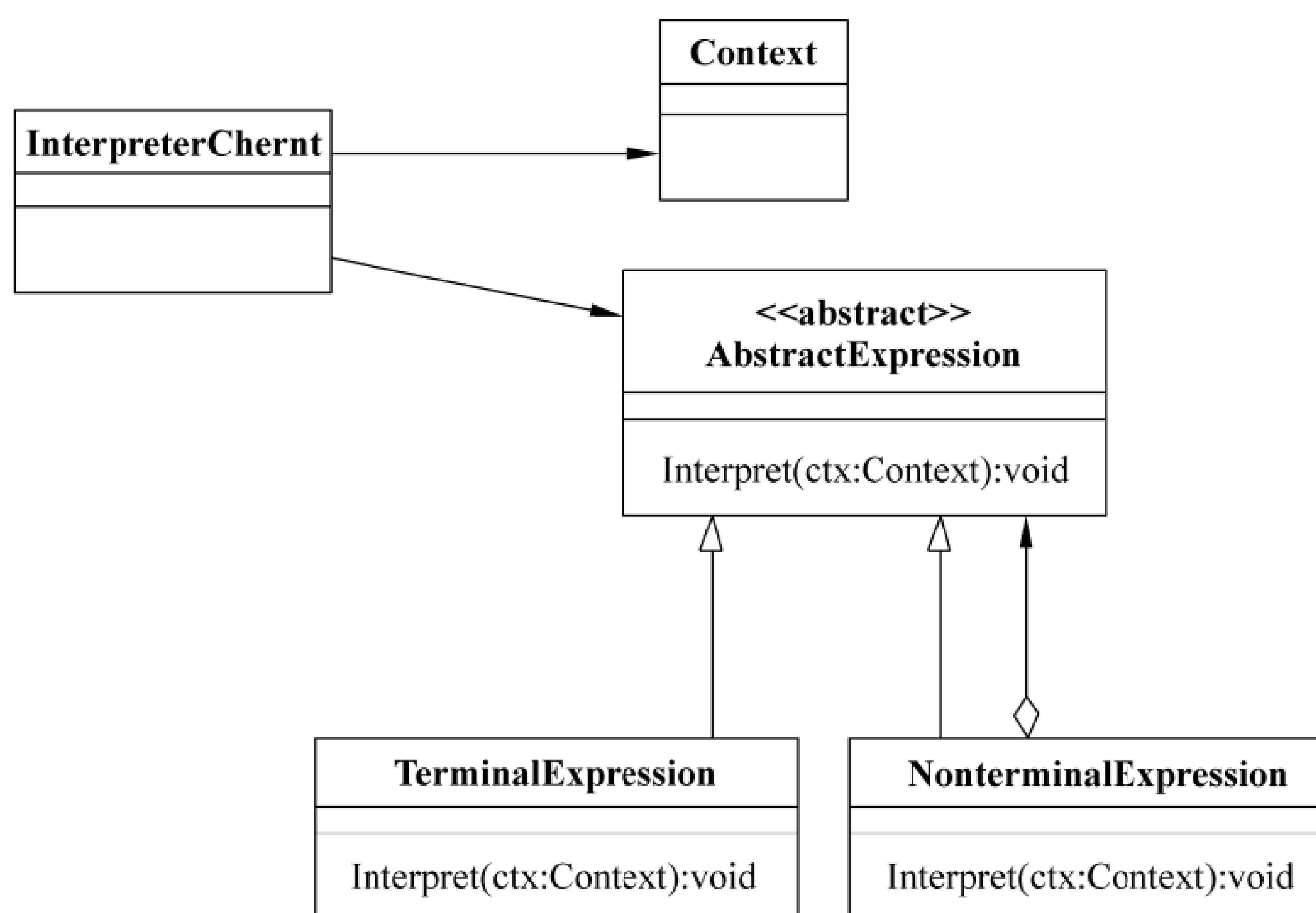


图 7-15 Interpreter 模式

其优点如下。

- 容易修改并扩展语法。
- 更容易实现语法。

在以下情况中，应该使用 Interpreter 模式：

- 语言的语法比较简单。
- 效率并不是最主要的问题。

4．Iterator 模式

Iterator 模式为集合中的有序访问提供了一致的方法，而该集合是独立于基础集合，

并与之相分离的。图 7-16 所示的是 Iterator 模式。

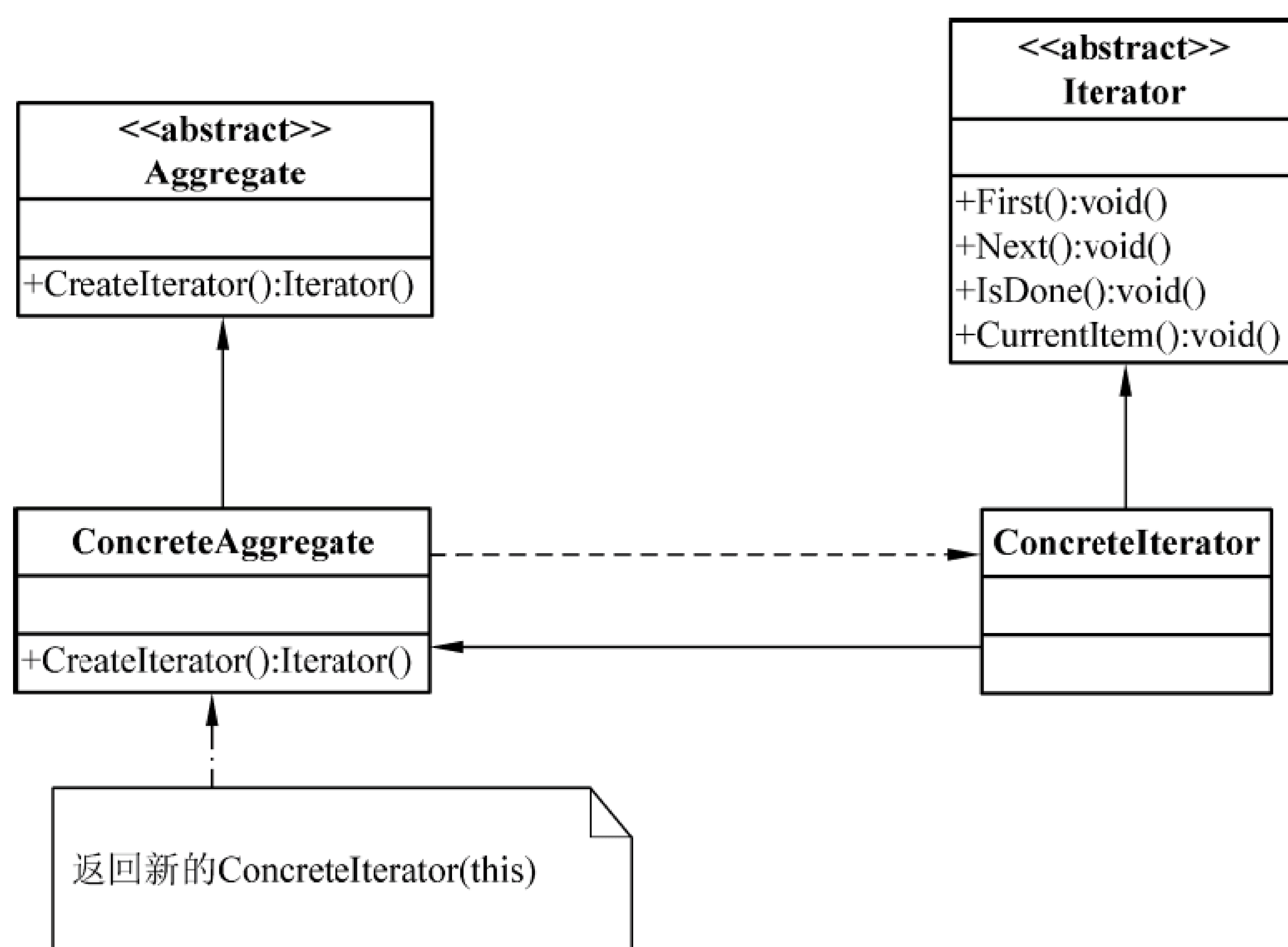


图 7-16　Iterator 模式

其优点如下。

- 支持集合的不同遍历。
- 简化了集合的接口。

在以下情况中，应该使用 Iterator 模式：

- 在不开放集合对象内部表示的前提下，访问集合对象内容。
- 支持集合对象的多重遍历。
- 为遍历集合中的不同结构提供了统一的接口。

5. Mediator 模式

Mediator 模式通过引入一个能够管理对象间消息分布的对象，简化了系统中对象间的通信。该模式可以减少对象之间的相互引用，从而提高了对象间的松耦合度，并且它还可以独立地改变其间的交互。图 7-17 所示的是 Mediator 模式。

其优点如下。

- 去除对象间的影响。
- 简化了对象间协议。
- 集中化了控制。
- 由于不再需要直接互传消息，单个组件变得更加简单，而且容易处理。
- 由于不再需要包含逻辑来处理组件间的通信，组件变得更加通用。

在以下情况中，应该使用 Mediator 模式：

- 对象集合需要以一个定义规范但复杂的方式进行通信。

- 想要在不使用子类的情况下自定义分布在几个对象之间的行为。

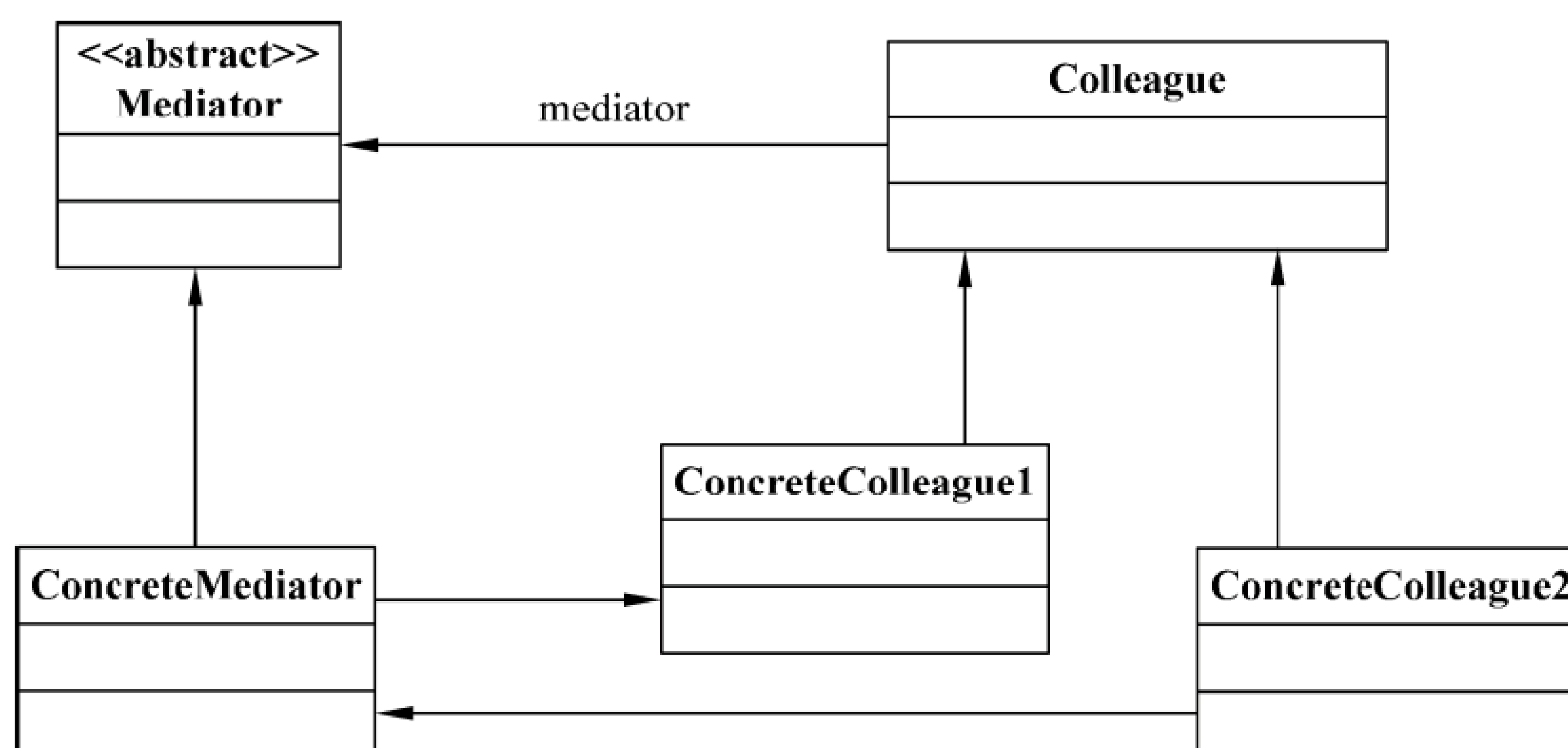


图 7-17 Mediator 模式

6. Memento 模式

Memento 模式可以保持对象状态的“快照”（snapshot），这样对象可以在不向外界公开其内容的情况下返回到它的最初状态。图 7-18 所示的是 Memento 模式。

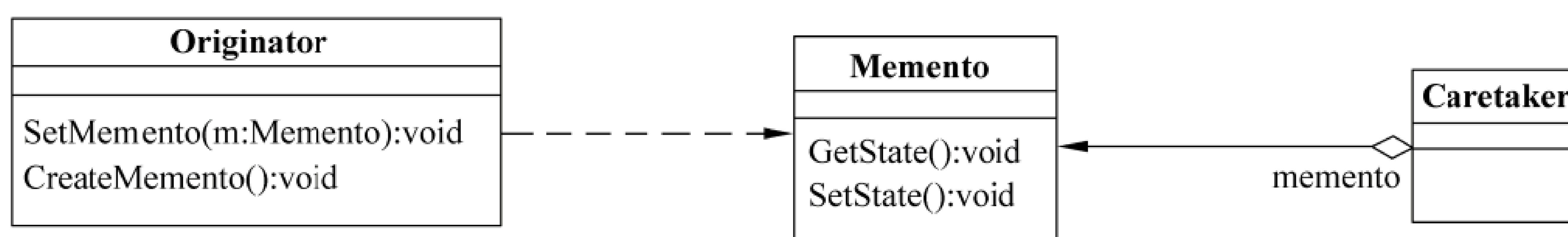


图 7-18 Memento 模式

其优点如下。

- 保持封装的完整。
- 简化了返回到初始状态所需的操作。

在以下情况中，应该使用 Memento 模式：

- 必须保存对象状态的快照，这样以后就可以恢复状态。
- 使用直接接口来获得状态可能会公开对象的实现细节，从而破坏对象的封装性。

7. Observer 模式

Observer 模式为组件向相关接收方广播消息提供了灵活的方法。该模式定义了对象间一到多的依赖关系，这样当对象改变状态时，将自动通知并更新它所有的依赖对象。图 7-19 所示的是 Observer 模式。

其优点如下。

- 抽象了主体与 Observer 之间的耦合关系。
- 支持广播方式的通信。

在以下情况中，应该使用 Observer 模式：

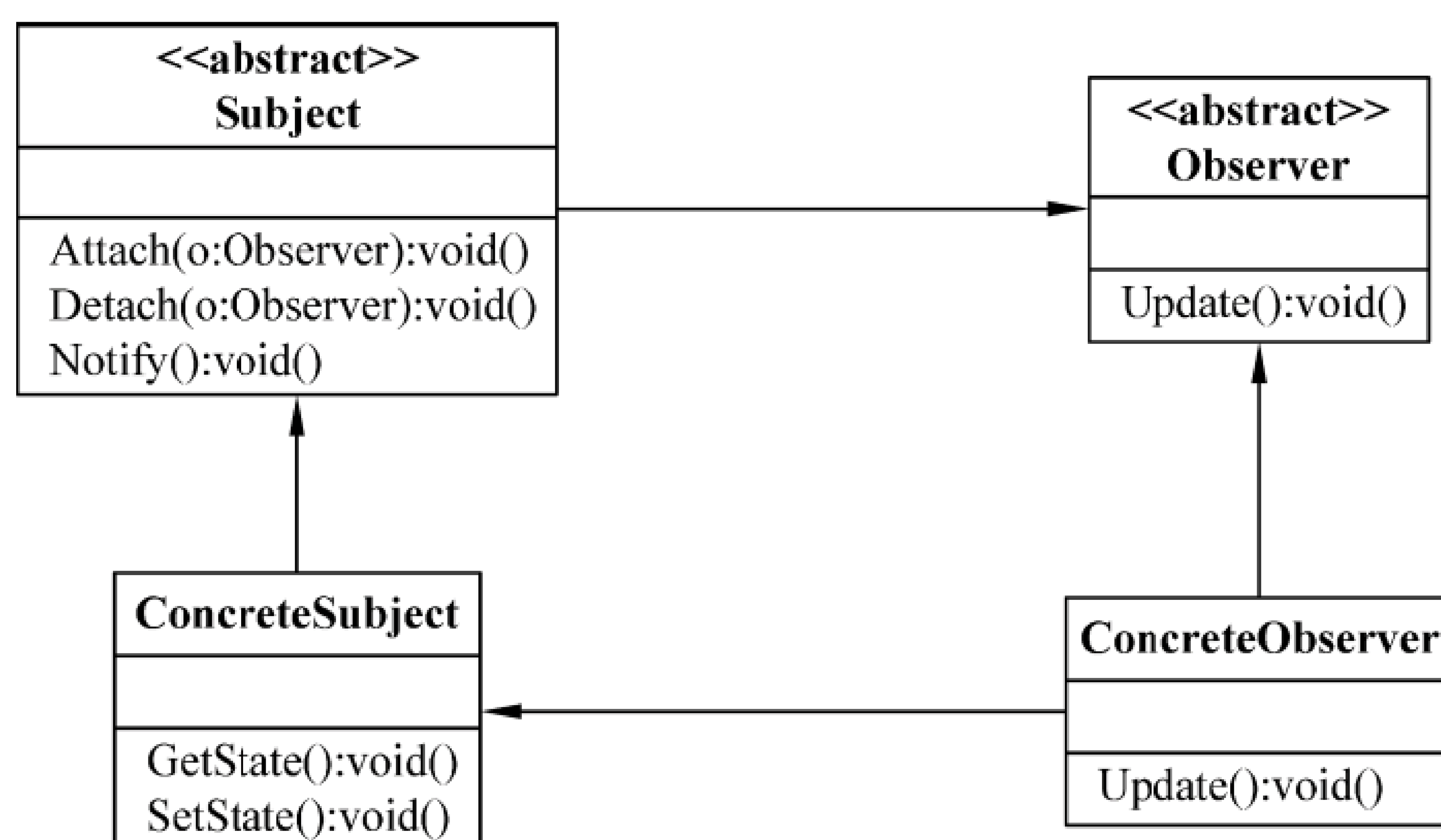


图 7-19　Observer 模式

- 对一个对象的修改涉及对其他对象的修改，而且不知道有多少对象需要进行相应修改。
- 对象应该能够在不用假设对象标识的前提下通知其他对象。

8．State 模式

State 模式允许对象在内部状态变化时，变更其行为，并且修改其类。图 7-20 所示的是 State 模式。

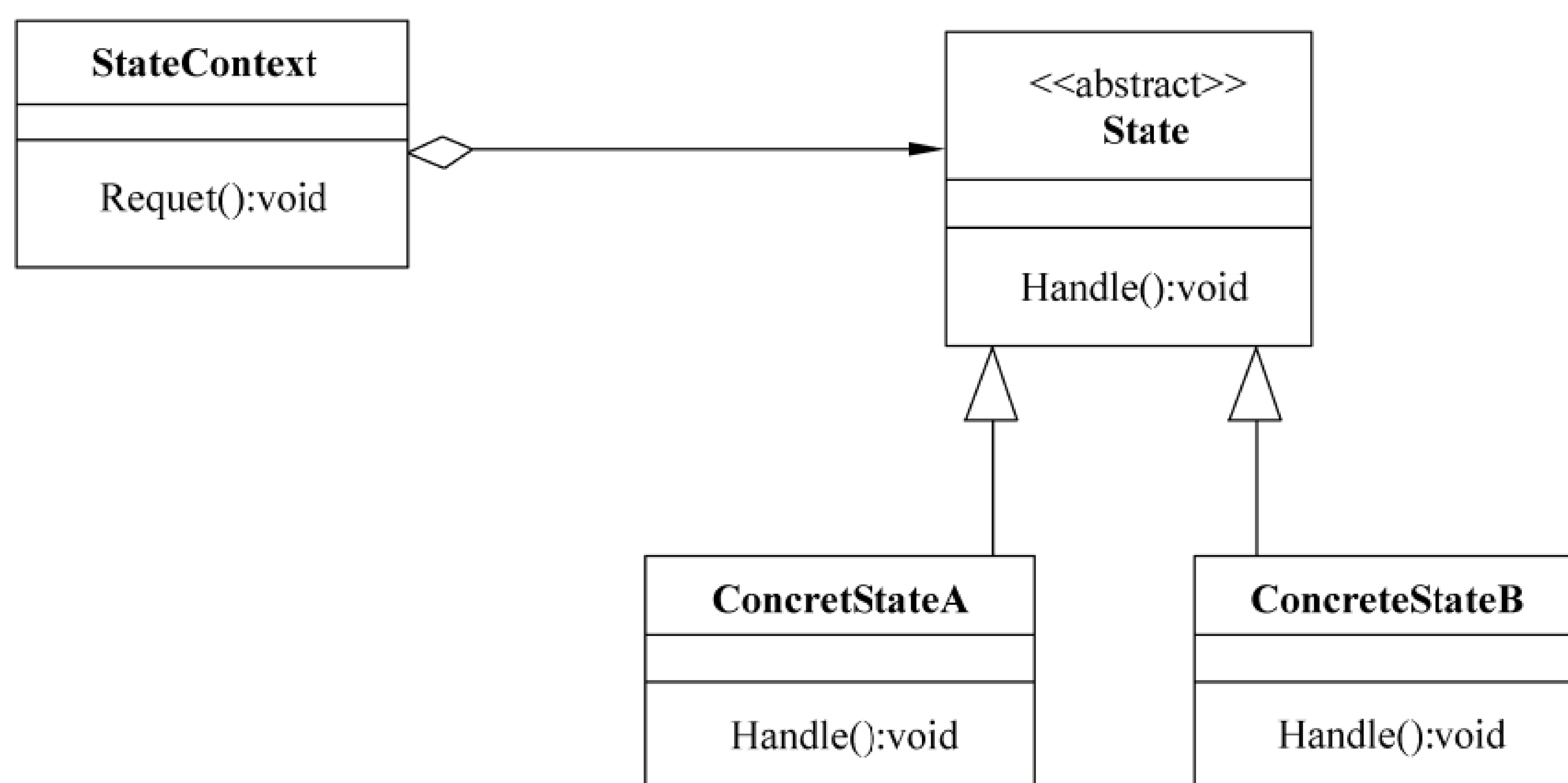


图 7-20　State 模式

其优点如下。

- 定位指定状态的行为，并且针对不同状态来划分行为，使状态转换显式进行。

在以下情况中，应该使用 State 模式：

- 对象的行为依赖于其状态，并且该对象必须在运行时根据其状态修改其行为。

- 操作具有大量以及多部分组成的取决于对象状态的条件语句。

9. Strategy 模式

Strategy 模式定义了一组能够用来表示可能行为集合的类。这些行为可以在应用程序中使用，来修改应用程序功能。图 7-21 所示的是 Strategy 模式。

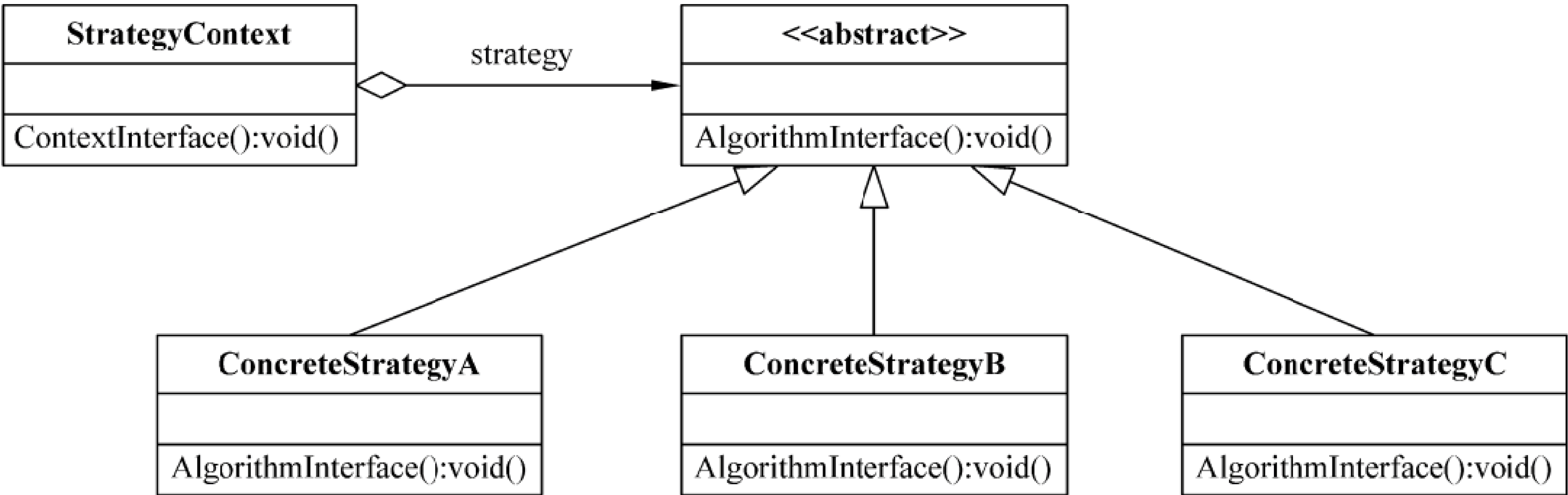


图 7-21 Strategy 模式

其优点如下。

- 另一种子类化方法。
- 在类自身中定义了每一个行为，这样就减少了条件语句。
- 更容易扩展模型。在不对应用程序进行代码修改的情况下，使该模式具有新的行为。

在以下情况中，应该使用 Strategy 模式：

- 许多相关类只是在行为方面有所区别。
- 需要算法的不同变体。
- 算法使用客户端未知的数据。

10. Template Method 模式

Template Method 模式提供了在不重写方法的前提下允许子类重载部分方法的方法。在操作中定义算法的框架，将一些步骤由子类实现。该模式可以在不修改算法结构的情况下，让子类重新定义算法的特定步骤。图 7-22 所示的是 Template Method 模式。

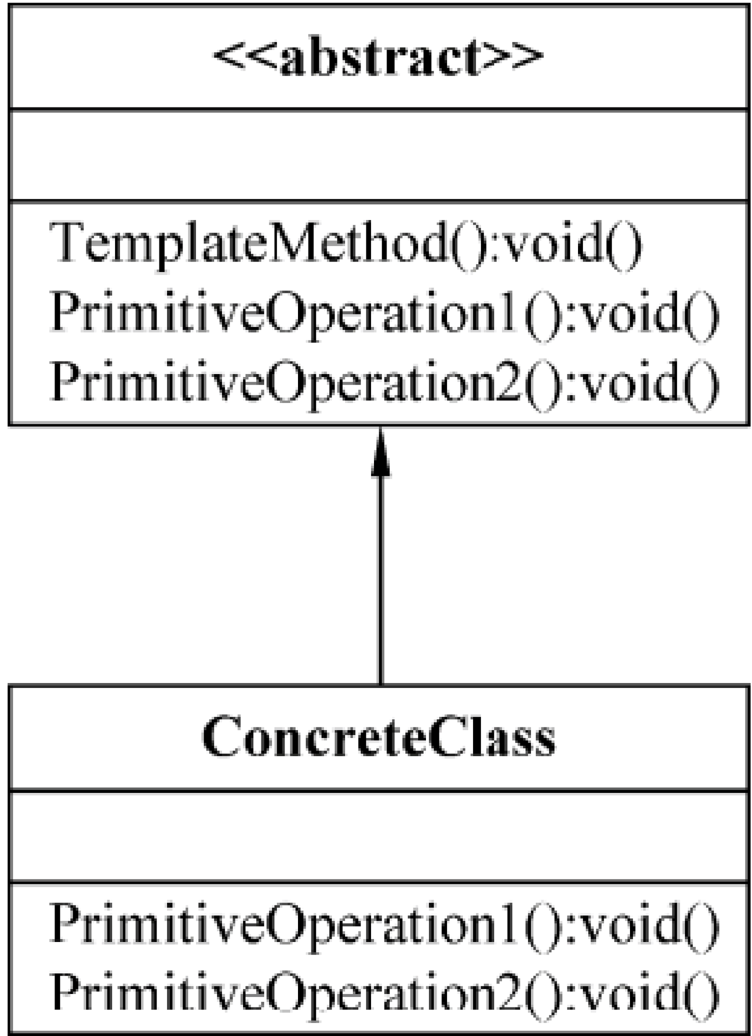


图 7-22 Template Method 模式

其优点为：代码重用的基础技术。

在以下情况中，应该使用 Template Method 模式：

- 想要一次实现算法的不变部分，而使用子类实现算法的可变行为。
- 当子类间的通用行为需要分解、定位到通用类的时候，这样可以避免代码重复的问题。

11．Visitor 模式

Visitor 模式提供了一种方便的、可维护的方法来表示在对象结构元素上要进行的操作。该模式允许在不改变操作元素的类的前提下定义一个新操作。图 7-23 所示的是 Visitor 模式。

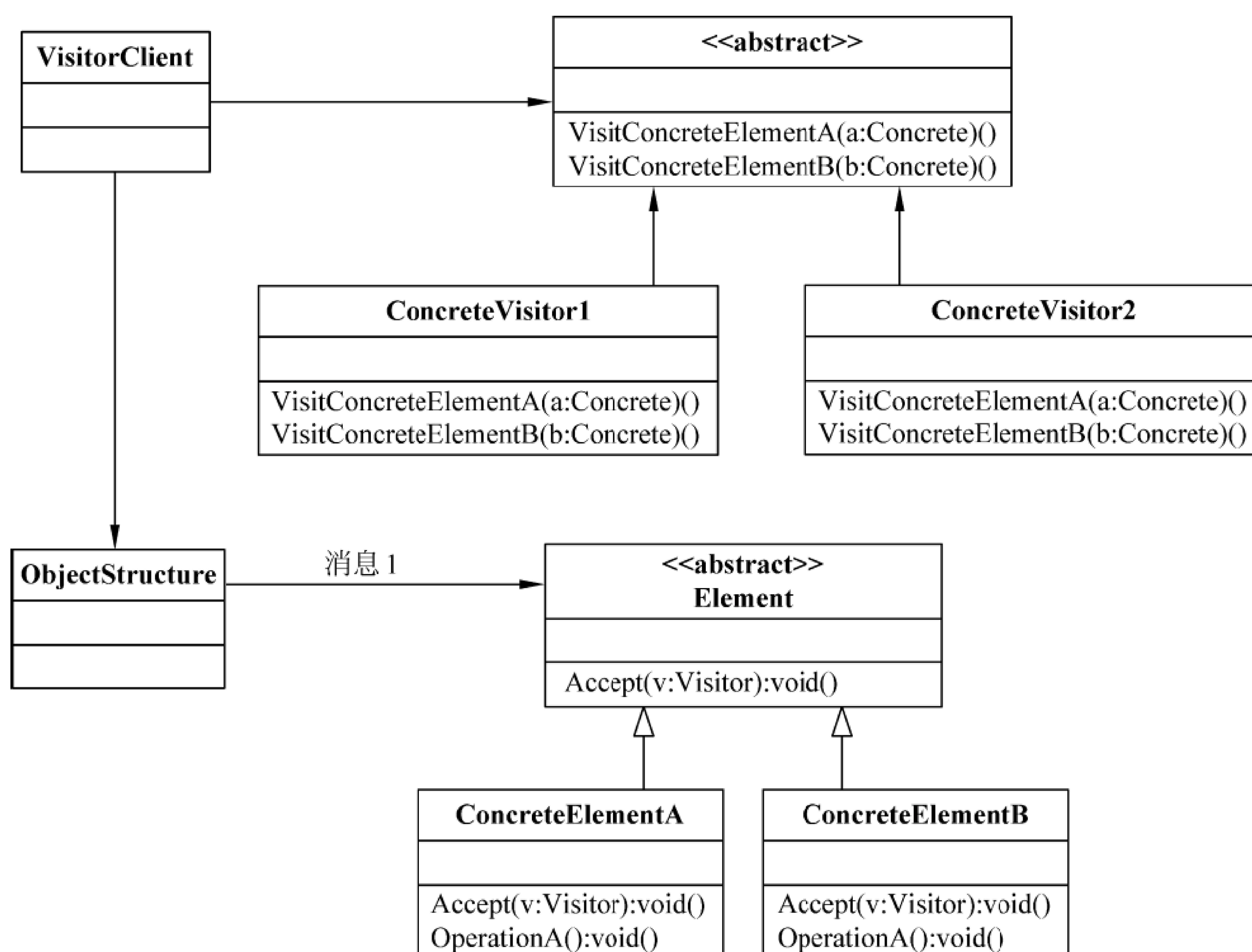


图 7-23　Visitor 模式

其优点如下。

- 更容易添加新操作。
- 集中相关操作并且排除不相关操作。

在以下情况中，应该使用 Visitor 模式：

- 对象结构包含许多具有不同接口的对象类，并且想要对这些依赖于具体类的对象进行操作。
- 定义对象结构的类很少被修改，但想要在此结构之上定义新的操作。

第 8 章　XML 技术

8.1　XML 概述

可扩展标记语言（XML）是标准通用标记语言（Standard Generalized Markup Language，SGML）的一个子集：XML 包含了很多 SGML 特性，但是比 SGML 要简单得多。像 SGML 一样，可以用 XML 来开发一种标记语言，它的元素和属性多是为专门行业和产业而定义的。创建这种语言之后，就可以使用 XML——就像使用 HTML（Hypertext Markup Language）一样——来标记并结构化文档了。

XML 和 HTML 都支持统一字符编码协会（Unicode Consortium）制定的通用字符集（Universal Character Set，UCS），它不仅包括特殊字符、标点和数学符号，还包括了非英语语言的字符和字母表，这使得 XML 成为了国际标准，这些字符和符号被认为是实体。XML 和 HTML 都支持样式表（style sheet）的使用，样式表有助于你定义整篇复杂的文档的结构和外观。然而，HTML 在许多方面都传统地定义了它的输出样式（当然最新版的 HTML 和由它继承而来的 XHTML 极力鼓励你使用层叠样式表来编制文档的样式）。而且，XML 在支持文档样式符号、规范语言（The Document Style Semantics and Specification Language，DSSSL）和层叠样式表（Cascading Style Sheets，CSS）等性能方面大大超过了 HTML。

专为 XML“服务”的样式表标准是可扩展样式表语言（Extensible Stylesheet Language，XSL），它是基于在线文档样式符号和规范语言（DSSSL-O）的，DSSSL-O 是 DSSSL 的一个子集，像 CSS 一样是专门为电子文档而创建的。有关 CSS 的更多信息，请参考同样由 Sandra E.Eddy 著 IDG Book Worldwide 公司出版的《XHTML 参考教程》，XSL 在本书第 2 部分的“XSL 样式表语法”和第 7 章的“使用 XSL 设计文档”中分别都有介绍。XSL 现在还只是一个建议，要跟上它的发展，请浏览其最新的建议：http://www.w3.org/TR/xsl/。XSL 转换（XSL Transformations，XSLT）是一种新的语言，它可以和 XML 协同工作。请参阅“XSLT 组件”中的有关 XSLT 的内容。

XML 包括如下特点和功能：

（1）XML 允许各种各样的文档显示类型，不仅可以显示在许多计算机平台上，而且可以显示蜂窝电话、掌上电脑等其他设备上。程序员可以使用任何编程语言或脚本语言来定义文档。

（2）XML 支持但并不需要 DTD（Document Type Definition，文档类型定义）。如果你使用 DTD，就要通知 XML 编辑器严格地按照 DTD 中设定的规则来处理文档。

（3）XML 的支持标准（XLink 和 XPointer）支持比 HTML 更为复杂的链接。在 HTML 和 XML 中你只能链接到一个 URL。然而在 XML 支持标准中可以同时使用几个或者一组链接。

（4）XML 标准支持打印文档和电子文档以及其他的为不同用户定义了不同内容和外观的文档。

（5）XML 支持客户端或者服务端计算机上的进程，这就允许开发人员分配资源和随时地节省资源。当前，XML 1.0 规范已经定义了，而 XLink 和 XPointer 语言仍处在开发和候选建议状态，这意味着它们已经被各技术团体评论过了。在成为最后建议之前，两种语言都可能发生改变。

8.1.1　XML 基本语法

本节将比较简略地介绍 XML 的基本语法，通过一个基本文档的例子来了解它的实质内容。希望这部分内容对于初次接触 XML 技术或者 Web 服务技术的读者，能有一个简要的知识铺垫。

以下是 XML 1.0 规范（第 2 版）的规范文本和该版本的中译本的 URL。

- http://www.w3.org/TR/2000/REC-xml-20001006
- http://lightning.prohosting.com/-ggiu/REC-xml-20001006-cn.html

8.1.2　标签语法

XML 标签负责提供、描述一个 XML 文件或数据包（也就是大家所熟知的 XML 实体）的内容结构。它们由界定内容的不同部分的标签（tag）所组成，负责提供到特殊符号和文本宏的引用，或者将特殊指令传递给应用软件，以及把注释传递给文档编辑器。

XML 元素的结构与 HTML 基本相同，XML 也同样使用尖括号来界定标签：以小于号（<）结尾，但二者的相同点也就仅此而已。

与 HTML 不同，几乎所有的 XML 标签都是大小写敏感的，其中包括元素的标签名和属性值，主要是满足 XML 国际化的设计目标和简化处理过程的需要。大多数非英语的语言并不把字母表分成若干种写法，许多字母可能也没有对应的大写或小写。合并写法会存在许多缺陷，尤其对于非 ASCII 码更是如此，而 XML 的设计者们大多选择避免这些问题。

1．字符

由于 XML 要在全球范围内使用，所以不能局限于 7 位的 ASCII 码字符集。XML 指定的字符均在 16 位的 Unicode2.1 字符集（参见 http://www.unicode.org，它目前与 ISO/IEC 10646 一致，后者可参见 http://www.iso.ch ）中定义。这些都是相对较新的标准，而且当今世界还有许多文字没有编入统一码中。但是，由于它被设计为大多数现存字符编码的超集，所以遗留的内容向统一码的转换也是简单直观的，例如，把 ASCII 码转换成统

一码只需要把 16 位字符的前 8 位填充为 0（而保留后 8 位）即可。

2．命名

在 XML 中使用的结构几乎总是被命名的。所有 XML 命名都必须以字母、下划线(_)或冒号（:）开头，后面跟着的是有效命名字符。有效命名字符除了前面的这些字符外，还包括数字、连字符（–）、句号（.）。在实际应用中不应该使用冒号，除非是用作命名空间修饰的分隔符（可参见本章后面的关于命名空间的相关描述）。字母并非局限于 ASCII 码，这一点是非常重要的，因为不说英语的人们可以把自己的语言用在标签中。

下面就是一些合法的命名：

Web、WEB、WebService：Interface、中国软件

注意前两个命名并不等同，因为 XML 的命名是大小写敏感的，第三个是使用建议的命名空间分隔符（冒号）的典型例子，最后一个例子提醒大家注意汉语同英语一样，都可以用于 XML 的命名。

下面是一些非法的命名：

-Web、4Web、Web$Service

8.1.3 文档部分

一个格式正规的 XML 文档由以下三个部分组成。

（1）一个可选的序言（prolog）。

（2）文档的主体（body），由一个或多个元素组成，其形式为层次树状结构，其中可能也包含了一些字符数据（character data）。

（3）可选的“繁杂”的尾声（epilog），其内容包括注释、处理指令（Processing Instruction，PI）和/或紧跟在元素树后面的空白。

8.1.4 元素

元素是 XML 标签的基本组成部分，它们可以包含其他的元素、字符数据、字符引用、实体引用、PI、注释和（或）CDATA（即 Charactor Data 缩写）部分——这些合在一起被称为元素内容（element content）（要注意这些元素都是容器）。所有的 XML 数据（除了注释、PI 和空白）都必须包容在其他元素中。

元素使用标签（tag）进行分隔：由一对尖括号（< >）围住元素类型名（一个字符串）。每一个元素都必须由一个起始标签和一个结束标签分隔开，这与要求比较松的 HTML 不同，后者的结束标签可以省略。这项规则唯一的例外是没有任何内容的元素，即空元素（Empty Element），它既可以使用起始标签/结束标签对，也可以使用短小精悍的混合形式——空元素标签。在后面，我们会看到许多标签的例子。

1．起始标签

一个元素开始的分隔符被称为起始标签。起始标签是一个包含在尖括号里的元素类

型名。我们也可以把起始标签看作是“打开”了一个元素，就像我们打开一个文件或通信链路一样。

下面就是一些合法的命名：

<Web>、<WEB>、< WebService: Interface>、<中国软件>

2. 结束标签

一个元素最后的分隔符被称为结束标签。结束标签由一个反斜杠和元素类型名组成，被围在一对尖括号中。每一个结束标签都必须与一个起始标签相匹配，我们可以把结束标签理解为“关闭”了一个由起始标签打开的元素。

下面是一些合法的结束标签，它们与前面列举的起始标签相对应。

</Web>、</WEB>、</WebService: Interface>、</中国软件>

所以，带有完整的起始、结束标签的元素应该是如下形式：

<某个标签>包含的内容</某个标签>

3. 空元素标签

空元素可能不包含任何内容。比如说想准确地指明文档中的某些特定位置，我们可以只加入起始标签和结束标签，而不在其中包含任何内容。

<WebService></WebService>

当然，如果你只是想指定一个点，而不是提供一个包容器，节省些空间可能会更好。所以，XML 指定空元素可以用缩略形式表示，它是起始和结束标签的混合体。它短小精悍，而且还能明确指出该元素既不会有内容，也不允许有内容。

空元素标签由一个元素类型名称紧跟一个反斜杠组成，并围在一对尖括号中。

<WebService/>

一个 XML 数据对象可能只包含单个文档根元素和一些空元素（可能有属性），这样的文件可以用来描述应用程序的配置信息或者面向对象编程语言中的对象模板。

4. 文档元素

格式正规的 XML 文档的定义形式是一个简单的层次树，每个文档都有一个，而且只有一个根节点，它被称为文档实体（document entity）或文档根（document root）。这个节点可能包含 PI 和（或）注释，而且总是包含子元素树，它们的根被称为文档元素（document element）。这个元素是这个树中其他所有元素的父元素，而且它可能不包含在其他任何元素当中。每个 XML 文档的文档根也是使用 DTD（Document Type Definition，文档类型定义）或模式定义的文档描述的附属品（由于本章并不想就 DTD 展开详细讨论，文章对于 XML 建模的重点是 XML Schema，因此只对 XMLSchema 进行讨论）。

任何格式正规的 XML 文档都必须由形成一个简单的层次树的元素所组成，其中有一个被称为“文档根”的单个根节点。它包含第二层的元素树，这个树也存在一个被称为“文档元素”的根节点。

5. 元素嵌套

XML 对元素有一种非常重要的要求：它们必须正确地嵌套。对现实世界对象的分析会有助于解释“正确嵌套”的含义。实际上，我们甚至可以说 XML 元素是任何必须遵守它们的现实来源规则的单词。

让我们来看一看本书传递到读者手中的整个过程。在完成印刷后，本书会和其他 23 本书打包到一个盒子中。两个盒子会被封装到一个纸箱中，许多纸箱会被装入一辆卡车然后运送到书店中。

整个过程可以用以下 XML 元素表示。

```
<trunk>
    <carton>
        <box>
            <book>…</book>
            <book>…</book>
            <book>…</book>
        </box>
        <box>
            <book>…</book>
            <book>…</book>
            <book>…</book>
            <book>…</book>
        </box>
    </carton>
    <carton>
        …
    </carton>
</trunk>
```

在上面的例子中，缩排只是为了突出这些嵌套元素的层次结构，为了简单起见也省略了许多对书和纸箱的描述。现实世界中的盒子能够包容整本书，但不可能出现书的某些部分在盒子中，而其他部分在外面的情况；同样，一本书也只能放在一个盒子中，不可能一部分在一个盒子，其他部分在另一个盒子。此外，盒子必须放在纸箱中，而纸箱必须顺序摆放在卡车里。当然，XML 元素也必须遵守这些现实世界包容关系的基本法则。

6. 字符串

字符串（string literal）主要用在属性值、内部实体和外部标识符中。XML 都使用单引号（' '）或双引号（" "）作为一对分隔符将其中的字符串包围起来。对于这些字符串的一个限制是用于分隔符的字符不能够出现在字符串中，如果字符串中包含单引号，分隔符就必须使用双引号，反之亦然。如果两个字符都必须出现在字符串中，用在字符串

中（同时也用作分隔符）的字符必须用适当的实体引用顶替（'或者" ）。

下面是一些合法的字符串表述：

'string'，"string". 'this's a "Web Service"'

而下面则是一些不合法的字符串表述：

"string'. 'this's a "Web Service"'

从技术的角度讲，根据 XML 规范，字符串分隔符之间的文本是文档字符数据的一部分。在讨论属性之前，我们先看一看它所包含的意义。

8.1.5　字符数据

字符数据就是任何不是标记的文本，它是元素或属性值的文本内容。小于号、大于号和&符号是标记分隔符，因此它们绝不能以字符串的形式出现在字符数据中（CDATA 部分除外，这一点我们将在后面提到)。如果这些字符是字符数据所必需的，它们必须使用实体引用“<”、“>”以及“&”来代替。这几个替代物是 XML 规范定义的 5 个类似字符串中的一部分，而且在所有兼容 XML 的解析器中都得到实现。

这里，需要再次提醒大家，由于 XML 的目的是在全球范围使用，所以文本是指统一代码，而不仅仅是 ASCII 码。现在，我们就来讨论属性的问题。

8.1.6　属性

如果说元素是 XML 中的名词，那么属性就是这种语言的形容词。在很多情况下，我们会希望将某些信息附着在元素上，它们与元素本身包含的信息内容有所不同。我们利用属性（attribute）来做到这一点，它们都包括一个名称/值对组合，使用的格式有如下两种形式：

```
attribute name="attribute value"
attribute name='attribute value'
```

属性值必须是分隔开的字符串（字符串规则的要求），其中可能包含实体引用、字符引用以及（/或）文本字符。但是，正如我们刚才解释的那样，任何一个受保护的标记字符（“>”、“<”和“&”）都不能简单地在属性值中当作字符使用，它们必须用“<”、“>”或“&”实体引用来替代。

HTML 允许数字化的属性，例如<IMGWIDTH=300>；或者不分隔的属性，比如<PALIGN=LEFT>；但这两种情况在 XML 中都不允许存在。

在起始标记或空标记中属性只允许有一个实例存在。例如，下面的例子在 XML 中就是非法的，因为 src 在一个标记中出现了两次：

```
<img src="abc.jpg" src="edf.jpg">
```

这种限制极大地简化了 XML 的处理。正如我们在前面暗示的，起始标记和空标记

可能在标记中包含属性。例如，回到我们前面提到的关于书本、盒子、纸箱和卡车的例子，如果我们希望给每个运送书本的纸箱编上一个号码的话，可以使用如下属性：

```
<carton number=" 0-232-93-1">
<carton number=' 0-232-93-2'>
```

在这个例子中，属性名称是 number，相应元素起始标签中的值为"0-232-93-1"及'0-232-93-2'。注意两个合法的字符串分隔符"和'在本例中都被使用了。

1．空白

不管是对于人类语言还是计算机语言来说，空白确实是一个非常重要的语言概念。在 XML 数据中，只有 4 个字符可以作为空白使用，如表 8-1 所示。

表 8-1　4 个字符的描述

字符值（十六进制）	描　　述	字符值（十六进制）	描　　述
09	水平指标（HT）	0D	回车（CR）
0A	换行（CF）	20	空格字符

无论如何，制表位占用的位置都不会只超过一个字符，所以它们中的每一个都可以简单地看作是一个字符。同样，任何由 LF 和（/或）CR 隐含的格式也是交给应用程序和（/或）样式表处理；同时，Unicode 定义了许多不同种类的空格，但其中没有一个能够成为 XML 中的空白。XML 处理空白的规则非常简单：解析器会保留内容中所有的空白字符并不加修改地传递给应用程序，但元素标记和属性值中的空白会被删除。

现在，让我们看一看 XML 是如何处理文档中的行尾的。

2．行尾的处理

XML 数据对象经常存储在离散的计算机文件中，它们被分割为若干个文本“行”。在 4 个 XML 空白字符中有两个是标准的 ASCII 码行尾控制字符。正如我们前面提到的，在用来表示行尾时，会有这两个字符的三种常见组合：CR/LF、只有 LF 以及只有 CR。

为了简化 XML 应用程序的编码，XML 解析器需要将所有的行尾字符串转换为单个 LF（换行）字符。很自然，这会让 Unix 编程者感到非常高兴，而让许多 MS-Windows 的开发人员怨声载道（Mac OS 用户已经适应了处理多种行尾字符串）。Tim Bray 曾经提出过一些折衷办法（主要是考虑到 MS-Windows 的市场份额），但结果是 XML 仍然要求使用 Unix 风格的行尾字符。

8.1.7　注释

这种机制对于在文档当中插入提示，或者叫注释（comment）来说是相当有帮助的。这些注释可能提供修订记录、历史信息或者其他类型的可能，这对创建者或者文档编辑者来说有着特殊意义，但又不是真正的文档内容的元数据。注释可能出现在文档中除其

他标记部分以外的任何地方。

XML 注释的基本语法如下。

```
<!--comment text-->
```

8.1.8　CDATA 部分

CDATA 部分是一种用来包含文本的方法，其对象是那些其中的字符如果不如此处理就会被识别为标记的文本。这项特性对于希望在自己的文档中包含 XML 标记的使用举例的作者来说是最有用的，就像本书中的举例。但这可能是在文档中包含 CDATA 部分的唯一说得过去的理由，因为在使用这些部分时 XML 几乎所有的优势都丧失殆尽。

只要有字符数据出现的地方就可能出现 CDATA 部分，但它们不能够嵌套。CDATA 部分的基本语法如下：

<![DATA[…]]>

在这里，"…。." 部分可以是任何字符串，只要不包含字符串 "]]>"。

由于在 Web 服务系列技术中，CDATA 同样不是常用技术，因此在这里也不加以详细讨论。

8.1.9　格式正规的文档

所有遵守 XML 语法规范的数据对象（文档）都是格式正规的 XML 文档。这类文档在使用时可以不使用 DTD 或模式来描述它们的结构，它们也被称作独立的（standalone）XML 文档。这些文档不能够依靠外部的声明，属性值只能是没有经过特殊处理的值或默认值。

一个格式正规（well-formed）的 XML 文档包含一个或多个元素（用起始和结束标记分隔开），它们相互之间正确地嵌套。其中有一个元素，即文档元素，包含了文档中其他所有的元素。所有的元素构成一个简单的层次树，所以元素和元素之间唯一的直接关系就是父子关系。兄弟关系经常能够通过 XML 应用程序内部的数据结构推断出来，但这些既不直接，也不可靠（因为元素可能被插入到某个元素和它的一个或多个子元素之间）。文档内容可能包括标签和（/或）字符数据。

数据对象如果满足下列条件就是格式正规的文档。

（1）语法合乎 XML 规范。

（2）元素构成一个层次树，只有一个根节点。

（3）没有对外部实体的引用，除非提供了 DTD。

任何 XML 解析器如果发现在 XML 数据中存在并不是格式正规的结构，就必须向应用程序报告一个“致命”错误。致命错误不一定导致解析器终止操作，它可以继续处理，试图找出其他错误，但它不再会以正常的方式向应用程序传递字符数据和（或）XML

结构。之所以采用这类错误处理方式，一是因为 XML 简洁的设计风格，二是因为 XML 更多的不是用于显示，因为这不太容易使得 XML 数据对象做到格式正规。

对于 HTML/SGML 来说，它们的工具都要比 XML 宽容许多。HTML 浏览器通常会显示出大多数支离破碎的 Web 页面，这为 HTML 的快速流行做出了巨大贡献。此外，真正的显示会因浏览器而异。同样，SGML（Standard Generalized Markup Language）工具即使遇到错误，通常也会尽力继续处理文档。

格式正规的文档的存在使得可以使用 XML 数据而不必承担构建和引用外部描述的重任。术语“格式正规”与正式的数学逻辑有着相似之处，一个命题如果满足语法规则就是格式正规，而不在于它的正确与否。

8.2 XML 命名空间

我们知道，XML 是一种元语言，我们可以使用 XML 来定义各种各样的应用。在上一节中，我们就已经看到了如此多的基于 XML 的规范，它们都是使用 XML 定义的 XML Application（XML 应用语言）。一般来说，我们可以使用 DTD 或者 XML Schema 来规范化定义每种特别的 XML。本书将不再介绍 DTD，如果有兴趣的话，可以阅读 XML 规范或者相关材料去了解其细节；对于 XML Schema，本书将在下一节结合实例进行描述。

无论是使用 DTD 还是 XML Schema，都是去定义一个专用 XML 词汇集以及使用这些词汇的规则，这样我们就不可避免地面对这样一些问题：

（1）如何知道我们在一个 XML 实例文档使用的 XML 词汇是在哪个 XML Application 中定义的？

（2）当我们混合使用两个 XML Application 的词汇集时，如果两个词汇集中有相同名字的元素名（当然它们表示的是不同的含义），如何区分它们？

同样，这些问题也会发生在我们自己来定义 XML 标签的场合中。比如说，如果你考虑使用 monitor 这样的元素，那么它在不同的环境将有几种不同的意思。如果你在计算机外围设备描述中使用，monitor 可能指的是计算机屏幕，同时在音乐制作间里扬声器通常也叫做 monitor。如果这里有一个专用于描述学校信息的数据模式，monitor 可能指的是一个被赋予几种职责的学生，然而在原子核电站，monitor 可能放在报警的地方。即便意思相同，在两种不同的定义中，其内容也会发生改变。

面对元素的这些潜在的不同用途，我们需要一种方法去区分元素的特定用途，特别是我们在同一个 XML 文档里混用不同的词汇。为了解决这个问题，W3C 提出了称为 XML 命名空间的规范，它允许我们在一个命名空间定义元素的前后联系，同时可以使用不同的命名空间来区分不同的 XML 词汇集中的元素名。

1999 年 1 月 14 日，XML 命名空间成为了 W3C 的推荐标准的程度。这一节将主要

介绍 XML 命名空间。命名空间帮助 XML 词汇表设计者去将复杂的问题分解成细小的问题，以及根据需要混合多义词来描述单一 XML 文档里的问题。模式允许词汇表设计者去建立更多而准确的词汇定义。

8.2.1　命名空间

XML 命名空间是解决多义性和名字冲突问题的方案。根据 W3C 组织的推荐标准 XML Namespace（1999 年 1 月 14 日）中的描述，XML 命名空间是一种名称的集合，通过一种 URI（Uniform Resource Identifier）引用来标识，作为元素类型和属性名称，它应用于 XML 文档。

命名空间是一组具有结构的名称的集合，这听起来像一个 DTD，的确，一个 DTD 可以是一种命名空间（因为一个 DTD 定义了一个 XML 词汇集）。在这种情况下，URI 可以是在你的服务器上的地址，如 http://www.uddi-china.org/schema/PubCatalog.dtd。

DTD 规定了一个文档的整体结构（并且是那么的准确），我们正好以一个命名空间为资源，规划所需要的定义。说到这里，一个命名空间不需要是一个像 DTD 那样的有固定结构的定义，而这个有限的定义领域使命名空间广泛应用于 XML。如果命名空间是 DTD 或者 XML Schema，我们使用的定义就必须在所描述的结构和语法上保持连续性。但是我们可以自由地使用需要的名称，并且使用命名空间来区分元素的使用。

于是，为了在文档里有效地使用命名空间，而文档中连接着来自不同地方的元素，我们需要两部分：

（1）URI 引用，定义了元素的使用方法。

（2）一个别名，我们可以用此来标识元素来自哪个命名空间，这将采用元素前缀的形式（例如在<中，catalog 是模糊的 contact 元素的命名空间别名）。

8.2.2　定义和声明命名空间

看到了命名空间在 XML 里所带来的优点，我们需要仔细看一下如何真正地使用它们。首先看一下在文档里怎样声明一个命名空间，然后看一下在文档里怎样使用命名空间，最后给出了几个例子。

通常，简单描述的特性作为属性来建模，并且这就说明了命名空间是怎样在 XML 中声明的。但这里有几个变形与转化，于是我们需要一步一步地去学习当在一个 XML 文档里声明一个命名空间时所能描述的东西。

1．声明一个命名空间

如果每个人在他们打算去识别一个命名空间声明的时候，我们需要在 XML 表示中提供一个保留的词汇给他们，命名空间推荐标准为我们提供了 xmlns 属性。属性值就是 URI，其唯一地定义了当前使用的命名空间。URI 经常是一个指向模式定义描述的 URL（可能是 DTD，也可能是 XML Schema 文档，甚至是其他）。用这种方式管理一个 URI，

以唯一区分命名空间已经足够了。

这里有几个简单的命名空间声明：

```
xmlns="http://www.uddi-china.org/schema/uddi_ v2.xsd"
xmlns="urn:catalog-specification"
```

关于 Web 资源的术语可能令人混淆。统一资源标识符（URI）是一些资源的唯一名称，它根据协议和网络位置定位资源。第一个例子是 URL，因为它允许一个浏览器利用 HTTP（Hypertext Transfer Protocolv）从一个特定的位置得到资源。第二个例子给资源命名但没提供位置，字面上的 urn 来自于 URI。

最初使用命名空间的动机之一是能够从不同的来源混合名称。基于这个考虑，为命名空间提供别名就成为非常有用的机制。而你能在一个涉及到相关声明的文档里通篇使用这个别名，可以靠加个冒号和你的别名到 xmlns 属性而实现该功能。因此上面的例子就变成了：

```
xmlns:uddi=http://www.uddi- china.org/schema/uddi_ v2.xsd
xmlns:catalog="urn:catalog- specification"
```

通过这样的命名空间声明之后，文档中的元素如果使用了 uddi 前缀（形如 uddi:businessEntity），那么表示这个元素使用了 http://www.uddi-china. org/schema/uddi v2.xsd 中描述的定义。而如果使用了 catalog 这个前缀（元素形如 ），那么则表明使用了 urn:catalog-specification 所标识的定义。使用这些声明和它们的别名为我们提供了更多的元素信息。当然如果没有前缀那就等同于使用了 xmlns="..."的命名空间约束。

2．修饰名

如果不能和一个我们想要使用的特定的名称绑定在一起，声明一个命名空间是没有什么用处的。这些已经通过利用修饰名（或成为命名空间修饰名，也就是使用命名空间前缀进行修饰的元素名或属性名）做到了。这就可能是你所希望的，一个从命名空间勾画出来并经其限定了的名称。通过别名创建一个确认过的名称，确切地说称作命名空间前缀，并把它放在名称的开始。下面是一个命名空间的例子：

```
xmlns:catalog=http://www.uddi-china.org/schema/catalog.xsd
```

此后我们就能够使用前缀 catalog 来进一步修饰每个元素名，使元素更加明确地表示出是来自哪个命名空间。于是，将要告诉我们 contact 名来自 catalog 命名空间声明；同样可能来自其他的命名空间，比如某个 Order 命名空间也有 contact 名称，但经过修饰的名称避免了多义性和冲突的可能性。

命名空间前缀经常被提及为前缀，而名称本身是基本名。修饰名可被应用于元素和属性名称。这里有一个混合一些命名空间的例子：

```
<catalog:contact identifier:id="30455716534">
```

这个元素从我们在上面看到的第一个命名空间那里产生，而属性 id 则从 identifier 命名空间产生。

3．作用范围

命名空间声明就像变量在程序语言里那样有它的作用范围。这非常重要，这是因为命名空间并不总是定义在 XML 文档开始，它们能够被包含在文档的较后部分。一个命名空间声明因此也应用于有声明出现的元素，尽管与此同时子元素并没有清清楚楚地描述出来。只要被用在命名空间声明的范围之内，就能够访问到命名空间。

但是我们也需要去混合命名空间，元素可能会回去另外地继承命名空间的作用域，于是这里有两种可以声明作用域的办法，即默认和修饰。

1）默认

如你所想象的，在一个文档里，在每一个名称前加一个前缀非常令人厌烦。实际上，通过在工具集里引入名称作用域的概念，能够分配很多前缀。如果定义了默认的命名空间（没有声明别名的，形式为 xmlns="..."），在声明作用域里所有没有经命名空间前缀修饰的名称被假定属于默认的命名空间。于是如果在根元素声明了一个默认的命名空间，它将被看作整个文档将默认的命名空间，并只能被在文档内部声明的其他的默认命名空间所覆盖。

通常省略前缀可以将一个命名空间声明为某范围内默认的。当一个前缀被定义并被一个名称引用时，明确地声明了命名空间。而对于那个带有命名空间声明但又不包含命名空间前缀修饰（使用默认命名空间）的元素，它们同样是属于其自身所带的命名空间声明所指定的命名空间的。

2）修饰

如果你能够清楚地区分命名空间，那当然非常好。但有些时候可能想要在一篇文档里从某个元素之外的命名空间来修饰名称，你就需要一个更精细的划分尺度。除了在整个空间声明命名空间，还可以利用带命名空间修饰的名称。在文档开头声明将需要的命名空间，然后在使用地点使用命名空间来修饰那些需要的元素和属性。

命名空间的存在主要是用来将名称组织成特有的集合以及回避名称冲突。W3C 命名空间推荐标准没有描述任何有关验证的使用方法。确实，XML 1.0 规范中没有说任何有关命名空间的东西。XML Schema 则将充分利用命名空间的功能。我们将在后面的章节中结合实例来阐述 XML Schema 的使用方法。

8.3　DTD

一个 XML 文档是有效的，则它必须满足：文档和一个文档类型定义或者模式相关联，这样 XML 处理器才能够理解并解析。那么怎样定义并利用这样一个文档类型定义

呢?本节会具体讲述文档类型定义（DTD）。

8.3.1 什么是 DTD

DTD 即文档类型定义，是 SGML、标准经 XML 的简化继承而来的，顾名思义，它主要是用来查看 XML 文档的格式，正如第 7 章所提到的，有关 DTD 的声明如果存在，则应出现在 XML 文档的序言中，以便 XML 校验器一开始就可以得到 DTD 所定义的该份 XML 文档的格式定义。另外，已经介绍过良构的 XML 文档和有效的 XML 文档的区别，所以可以知道 DTD 声明不是必须出现的。

在 DTD 中主要定义以下几个方面的内容：

（1）元素声明，包括元素的内容和元素的排列组合方式。

（2）实体声明。

（3）属性的种类。

8.3.2 为什么引入 DTD

首先考虑一下 DTD 的作用问题。事实上，DTD 也好，XML Schema 也好，它们都是提供一种验证的手段。验证对于 XML 来说是一大贡献，正是有了验证才可以确保 XML 文件确实地遵守了指定的格式，而这个格式可能是一个标准，或是数据交换双方所共同制定的协议，也正是如此，XML 在电子商务领域掀起了轩然大波。

DTD 的使用实现了文件格式的统一化，促进了行业或系统内部的文件的标准化，同时也提高了文件的重用性。DTD 定义了文件的格式，这种统一的格式可能为某一领域的大量文件所共用，减少了文件制作的时间。也正是由于 DTD（和 XML Schema）的存在才突出了 XML 文档的数据的结构性。

但是也应该看到，并不是所有的文件都强调结构性。对于本身的结构性很强，或是在实际应用中不强调它的结构性的文档，使用 DTD 进行验证，增加了操作时间，得不偿失，可以考虑不使用 DTD，只建立良构的 XMI 文档就行了。

8.3.3 DTD 的声明

1. 内部 DTD 声明

DTD 的声明分为内部 DTD 声明和外部 DTD 声明。在本节通过如下例子介绍内部 DTD 声明。

```
<?xml version="1.0" encoding="UTF-8"?>
< !DOCTYPE Transaction[
< !ELEMENT ContractDate(#PCDATA)>
< !ELEMENT ContractName(#PCDATA)>
< !ELEMENT ContractNo(#PCDATA)>
```

```
< !ELEMENT ContractType(#PCDATA)>
<!ELEMENT Transaction(ContractNo, ContractType, ContractName,
ContractDate)>]>
< Transaction>
    <ContractNo>ContractNumber</ContractNo>
    <ContractType>FOB</ContractType>
    <contractName>ContractName</ContractName>
    <ContractDate>ContractDate</ContractDate>
</Transaction>
```

从上面的例子中可以看到内部 DTD 声明的语法是：

< !DOCTYPE 根元素名称[...DTD 规则

…]>

可以看出内部 DTD 声明是内嵌于相应的 XML 文档当中的，这种方式虽然简单，但是从引入 DTD 的原因考虑，很容易想到，这种做法并不利于文档结构的重用。为此 XML 还提供了另一种 DTD 声明——外部 DTD 声明。

2．外部 DTD 声明

外部 DTD 声明的语法规则可以描述为如下形式。

<!DOCTYPE 根元素名称 SYSTEM DTD 的 URI>或者

<!DOCTYPE 根元素名称 PUBLIC DTD 的名称 DTD 的 URI>

可以看出，外部 DTD 声明是依照两种方式进行的，分别把它们叫做 SYSTEM 方式和 PUBLIC 方式。下面分别说明两种方式的使用方法。

1）SYSTEM 方式的外部 DTD 声明

与 PUBLIC 相对，使用 SYSTEM 模式的 DTD 一般在未公开、私人或小团体内使用，可以根据有关 URL 的规定设置该 DTD 与用它的 XML 文档之间的路径关系。

- “..” 代表到该文档所在目录的上层目录。
- “/” 代表子目录。
- 默认情况代表当前目录。

下面是一个 XML 文档，它使用的是 SYSTEM 方式的外部 DTD 声明。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
< !DOCTYPE Transaction SYSTEM "dtd1.dtd">
< Transaction>
    <ContractNo>ContractNumber</ContractNo>
    <ContractType>FOB</ContractType>
    <contractName>ContractName</ContractName>
    <ContractDate>ContractDate</ContractDate>
</Transaction>
```

这个 XML 文档引用的 DTD 文件 dtd1.dtd 内容如下：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
< !ELEMENT ContractDate(#PCDATA)>
< !ELEMENT ContractName(#PCDATA)>
< !ELEMENT ContractNo(#PCDATA)>
< !ELEMENT ContractType(#PCDATA)>
<!ELEMENT Transaction(ContractNo, ContractType, ContractName,
ContractDate)>]>
```

关于这个例子，需要说明的是，在处理指令<?xml version="1.0" encoding="UTF-8" standalone="no"?>时，属性 standalone 应该设置为 no，因为此时不是单个的 XML 文件，还需要读取它的 DTD 文件。

2）PUBLIC 方式的外部 DTD 声明

在很多情况下，可能与其他合作伙伴共同制定为一个行业或某个工作领域共同遵守的 XML 文档格式，这时所使用的 DTD 就不是小团体或个人所属的，而具有公共性质，就如本书下半部分将要介绍的电子商务领域的 XML 标准。这时作为一种公用的 DTD，它可能存放在某个 Web 网站上，用 PUBLIC 方式的外部 DTD 声明指明这种公用的 DTD。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
< !DOCTYPE Transaction PUBLIC TransactionDTD "http://202.116.64.1/
dtd1.dtd">
< Transaction>
    <ContractNo>ContractNumber</ContractNo>
    <ContractType>FOB</ContractType>
    <contractName>ContractName</ContractName>
    <ContractDate>ContractDate</ContractDate>
</Transaction>
```

关于这个例子，需要说明的是，DTD 的 URL 必须是绝对路径，另外可能注意到了增加了一个参数"TransactionDTD"，这个参数是 DTD 的名称，如果公用 DTD 的原作者没有指明它的名称，可以自行定义一个名称。

事实上，XML 的验证器在验证 XML 文档是否符合它所声明的 PUBLIC 方式的外部 DTD 声明时，首先将外部 DTD 从指定的 URL 处下载，然后做与 SYSTEM 方式下同样的验证。

3．内部 DTD 和外部 DTD 同时使用

DTD 的使用方式相当的灵活，它不仅提供了内部 DTD 和外部 DTD 两种方式，而且还支持内部 DTD 和外部 DTD 的同时使用以及和多个外部 DTD 的同时使用，那么很自然想到一个问题，当两种 DTD 有一部分声明是重复的，该怎么办呢？事实上，在两种

声明存在重复的情况下，处理重复部分的方式和面向对象概念中的重载很类似，把内部声明看作对外部声明中重复部分的再一次说明，重复的部分以内部声明为主。这样就大大提高了外部声明的重用率，使得取用外部 DTD 的机会增加。

8.3.4　元素的声明

元素是 XML 文档中相当重要的组成部分，通过介绍在 DTD 中元素的声明方式进一步了解元素。

元素声明的基本语法可以写作：

<!ELEMENT 元素名称 元素定义>

其中 ELEMENT 是 XML 中预留的关键字。可以看出在元素声明中比较复杂的是元素定义部分。下面分不同的情况介绍元素的定义。

1．空元素的声明

空元素应该说是最简单的元素，它通过预留关键字 EMPTY 来实现。下面是一个空元素的声明：

<!ELEMENT 附加条件 EMPTY>

2．文本元素的声明

这里主要指元素的内容完全是文本形式的元素类型，XML 中用（#PCDATA）来描述，意思是可解析数据。下面是一个文本元素的声明：

<!ELEMENT 地址（#PCDATA）>

3．无限制元素的声明

这里所指的无限制元素是 XML 为某些结构性比较差的元素提供的说明方法，它对该元素不作限制。这里的不作限制可以理解为：

（1）可以包含任何在该 DTD 中声明过的子元素。

（2）这些子元素的出现次数与次序不作限制。

（3）可以包含文本数据，即（#PCDATA）型数据。

（4）数据内容与子元素可以交错出现。

XML 通过关键字 ANY 实现对无限制元素的声明。这类无限制元素的声明语法如下：

< !ELEMENT 附加条件 ANY>

4．包含子元素的元素声明

第 7 章就介绍过了关于根元素、父元素和子元素的概念。事实上，XML 文档本身通过元素之间的嵌套关系描述了数据（元素）之间树状结构关系，正因如此 DOM 等应用程序接口才能十分方便地将 XML 文档转化为许多应用程序习惯处理的树状结构或其变种。

所以，考虑包含子元素的元素声明时，应该注意不只要定义包含的子元素是什么，而且要定义这些子元素出现的次数和次序，在 XML 中通过一些结构符号描述这些信息，

这些符号的描述如表 8-2 所示。

表 8-2　结构符号的具体描述

<table>
<tr><th>结构符号</th><th>使用方法</th><th>应用实例</th><th>实例说明</th></tr>
<tr><td>()</td><td>将括号内的元素或数据类型合并为一个单位</td><td>（A，B，C）</td><td>A、B、C 可能是元素或#PCDATA 数据，将 A、B、C 合成一个单位</td></tr>
<tr><td>、</td><td>元素或数据以出现的顺序排列</td><td>（A，B，C）</td><td>元素 A、B、C 以出的顺序排列</td></tr>
<tr><td>*</td><td>该元素或是由括号形成的单位出现任意次数，包括 0 次</td><td>A*
（A，B）*</td><td>前一例表明元素 A 可能出任意次；后一例表明元素 A 和 B 以一前一后的次序出现任意次</td></tr>
<tr><td>+</td><td>该元素或是由括号形成的单位出现任意次数，不包括 0 次</td><td>A+
（A，B）+</td><td>前一例表明元素 A 可能至少出现一次；后一例表明元素 A 和 B 以一前一后的次序至少出现一次</td></tr>
<tr><td>?</td><td>该元素或是由括号形成的单位出现一次，即 0 次或 1 次</td><td>A?</td><td>元素 A 出现 0 或 1 次</td></tr>
<tr><td>|</td><td>在该元素范围内的元素只能并必须出现一个</td><td>（A|B|C）</td><td>元素 A 出现或 B 出现或 C 出现</td></tr>
</table>

8.3.5　实体的声明

1．什么是实体

简单地说，实体（entity）是一些预先定义好的数据，在需要使用这些数据的时候通过引用的方法将它放入特定的位置。使用实体的好处在于对于一些重用率很高的数据，通过将它们定义成实体，减少对这类数据进行修改是必须做的重复工作。另外，通过实体可以把一些特殊的数据，如声音文件、图片文件等引入 XML 文档。

根据不同的角度可以对实体进行多种不同的分类：按照实体存储的部位的不同，实体可以分为内部实体和外部实体；按照实体的组成内容的不同，实体可以分为可分解实体和不可分解实体两种；按照实体引用方式的不同，可以分为一般型实体和参数型实体。不同类型的实体的声明和使用方法略有不同，下面进行详细的介绍。

2．内部实体和外部实体

内部实体在声明的时候就被完整地定义，因此应该想到，内部实体只能是文本型的数据。内部实体的声明方法如下：

<!ENTITY 实体名称 实体内容>

实体的引用方法是：

&实体名称;

如下例子说明如何声明并引用实体。

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
< !DOCTYPE Transaction[
< !ELEMENT ContractDate(#PCDATA)>
< !ELEMENT ContractName(#PCDATA)>
< !ELEMENT ContractNo(#PCDATA)>
< !ELEMENT ContractType(#PCDATA)>
<!ELEMENT Transaction(ContractNo, ContractType, ContractName,
ContractDate)>]>
< Transaction>
    <ContractNo>ContractNumber</ContractNo>
    <ContractType>FOB</ContractType>
    <contractName>ContractName</ContractName>
    <ContractDate>ContractDate</ContractDate>
</Transaction>
```

外部实体的声明虽然在 DTD 文档中，但是实体代表的数据本身存储于 DTD 文档之外。外部实体的声明方法如下：

<!ENTITY 实体名称 SYSTEM I PUBLIC "URI">

有关属性 SYSTEM 和 PUBLIC 的意义和 DTD 声明中介绍的一样，这里不再重复。外部实体的引用方法和内部实体是一样的。

最后要说明的是特殊字符的使用问题。在 XML 中，一些字符被赋予了特殊的意义，当不想使用这些字符的特殊意义而引用它们时，就使用实体引用的方式。表 8-3 是常见特殊字符的实体引用方式。

表 8-3　常见特殊字符的实体引用方式

特殊字符	特殊字符的引用	特殊字符	特殊字符的引用
<	“&.lt;”	'	“&.apos;”
>	“&.rt;”	"	“&.quoot;”
&	“&.amp;”		

3．可分解实体和不可分解实体

可分解实体又称为文本实体。所谓可分解是指可以被 XML 验证器所解读，并且在解读后将实体的内容放入引用该实体的位置上去，例如相对于可分解实体，不可分解实体就是不能直接被 XML 验证器所解读的实体，例如图片、声音文件等。由于 XML 校验器本身不能解读这种实体，所以在声明这类实体时应该声明它的格式以及处理该实体的应用程序的位置。下面是不可分解实体的声明方法：

<!NOTATION 格式 SYSTEM|PUBLIC 处理程序的 URI>

<!ENTITY 实体名称 SYSTEM|PUBLIC 实体的 URI NDATA 格式>

实体的引用方式没有变化。

```
<?xml version="1.0" encoding="UTF-8"?>
< !DOCTYPE Transaction[
< !ELEMENT ContractDate(#PCDATA)>
< !ELEMENT ContractName(#PCDATA)>
< !ELEMENT ContractNo(#PCDATA)>
< !ELEMENT ContractType(#PCDATA)>
<!ELEMENT Transaction(ContractNo, ContractType, ContractName,
ContractDate)>
<!NOTATION gif SYSTEM "C:\paint.gif">
<!ENTITY ProductPicture SYSTEM "picture.gif" NDATA gif>]>
< Transaction>
    <ContractNo>ContractNumber</ContractNo>
    <ContractType>FOB</ContractType>
    <contractName>ContractName</ContractName>
    <ContractDate>ContractDate</ContractDate>
<Product>
      <Picture>&ProductPicture</Picture>
</Product>
</Transaction>
```

4．一般型实体和参数型实体

到现在为止，所介绍的实体都是一般型实体。这种类型的实体不仅可以在 XML 文件本体中引用，而且可以在它所在的 DTD 中使用。在所在 DTD 中使用应该注意的是，一般只用于对另外的实体的声明，不用于对元素的声明中，同时应该避免实体中引用实体时产生循环引用的现象。

一般型实体在引用时都采用如下方式：&实体名称；而将要介绍的参数型实体则有很大不同，这类实体只在外部 DTD 声明中使用，而且不同于一般型实体，它可以用于元素的声明中。它的声明语法如下：

<!ENTITY % 实体名称 实体内容>

引用语法如下：

% 实体名称

可见，参数型实体和一般型实体有着很大的不同。事实上，可以把参数型实体看作在声明外部 DTD 时为避免重复工作提供的一种方法。如果有机会读到一些用作行业标准的 DTD 文档，会发现其中存在着大量的参数型实体的引用。

对一般型实体和参数型实体的区别总结如表 8-4 所示。

表 8-4 一般型实体和参数型实体的区别

项 目	一般型实体	参数型实体
声明语法	<! ENTITY 实体名称 实体内容>	<! ENTITY%实体名称 实体内容>
声明处	外部 DTD 与内部 DTD 都可以	只能在外部 DTD

续表

项　目	一般型实体	参数型实体
实体引用语法	&实体内容	%实体内容
实体引用限制	元素声明中不能用	外部 DTD 和 XML 文件本体中不能用
实体引用的适用处	DTD 实体的声明处；XML 文件本体内	外部 DTD 的实体声明和元素声明处

8.3.6　属性的声明

属性是 XML 提供的描述元素某些性质的信息，XML 本身提供了一些默认属性，如最早在处理语句<?xml version="1.0" encoding="UTF-8" standalone="no"?>就接触到默认属性 version、encoding 和 standalone。当然除了默认属性以外，还可以根据需要自行设计属性，在一个良构 XML 文档中，属性只要满足命名规则就可以了，但是在一个有效的 XML 文档中，属性要经过 DTD 的属性声明，这是 DTD 声明中比较复杂的一部分。

在 DTD 声明中，属性的声明语法可以归纳为如下形式。

<! ATTLIST 元素名称 属性名称 属性类型 属性默认值类型>

这里需要说明的是，元素名称指的是属性所属的元素名称，关于属性类型和属性默认值类型是属性中比较复杂的内容，下面详细地进行介绍。

1. 属性类型

属性类型是对属性取值的约束，属性值类型共有 7 种选择，下面一一介绍。

（1）CDATA 类型：CDATA 类型是最简单的属性类型，也是约束最少的属性类型，代表该属性值为一般的文字，除此没有其他限制。声明语法如下：

<! ATTLIST 元素名称 属性名称 CDATA 属性默认值类型>

（2）枚举型：枚举型列举了该属性所能取得的全部值。声明语法如下：

<! ATTLIST 元素名称属性名称（可选属性值 1|可选属性值 2|...|可选属性值 n）属性默认值类型>

（3）NMTOKEN 和 NMTOKENS 类型：使用 NMTOKEN 属性类型时，表明该属性的属性值只能由字母、数字、“_”等字符所构成的字符串，即属性值满足命名规则，且不能出现空格。NMTOKENS 属性则表明属性值可能由若干个满足 NMTOKEN 属性要求结合在一起形成的，即多个 NMTOKEN 之间可能存在空格，声明语法如下：

<! ATTLIST 元素名称 属性名称 NMTOKENINMTOKENS 属性默认值类型>

（4）ENTITY 和 ENTITIES 类型：当属性值是一个外部实体的引用时，用 ENTITY 来说明属性类型；一般来说，当属性值是一个内部实体引用时，将属性类型声明为 CDATA 即可。

（5）NOTATION 类型：在介绍不可分解实体类型时已经提到过 NOTATION 的声明，当属性只是一个声明过 NOTATION 格式时，将属性声明为 NOTATION 类型。由此也可

以看出，在把一个属性声明为 NOTATION 类型时，首先应该确定存在相应的 NOTATION 声明。下面把 NOTATION 声明和 NOTATION 类型属性声明的语法写在一起。

<NOTATION 格式 SYSTEM I PUBLIC 处理程序的 URI>

<ATTLIST 元素名称属性名称 NOTATION 属性默认值类型>

（6）ID 类型：CDATA 类型对于属性的限制是比较少的，ID 类型是在 CDATA 类型的属性类型上加入一点更强的约束条件，那就是作为此类型值的名字只能在 XML 文件中出现一次。此即，ID 类型的值必须能唯一标识元素。

（7）IDREF 和 IDREFS 类型与 ID 类型相对，IDREF 就是 ID REFERENCE 的意思，表明该属性的取值是对声明过的一个 ID 型属性值的引用，也就是说，IDREF 类型的值必须匹配某些 ID 属性的值。同样道理，IDREFS 类型表明该属性的取值是对声明过的多个 ID 型属性值的引用。

2．属性默认值类型

属性默认值类型有 4 种可选情况，下面一一介绍它们的使用时机和使用方法。

（1）# IMPLIED 表明该属性可出现可不出现，声明语法：

<！ATTLIST 元素名称 属性名称 # IMPLIED>

（2）# REQUIRED 表明该属性一定要出现，声明语法：

<！ATTLIST 元素名称 属性名称 # IMPLIED>

（3）#FIXED 表明该属性一定要出现，且固定为特定值，不许用其他值，声明语法：

<！ATTLIST 元素名称 属性名称 # FIXED 属性特定值>

（4）特定属性值表明该属性的默认值，当 XML 文档中没有显式给出该属性的取值时，取此值。当该属性的取值已经显式地给出时，则为给出值：

<--! ATTLIST 元素名称 属性名称 属性默认值>

8.4　XML Schema

DTD 在制定标记语言方面历史悠久，早在 XML 正式标准出现以前就已经存在，当时它配合 SGML 来制定标记语言，是专门为 SGML 服务的 DTD。当 XML 出现之后，DTD 尽管进行了很大的简化，但还是一门风格和 XML 完全不同的语言，不经过细致学习，根本不可能应用这种语言制定标记语言。而 schema 文档是一种特殊的 XML 文档，它遵循 XML 的语法要求，避免用户再去另外学习一套语法，同时 schema 语法结构简单，容易学习和使用。因而在发展势头上 schema 远高于 DTD。

DTD 的另一个缺点是数据类型相当有限。在 DTD 中根本不提供数值数据类型，例如整数、浮点数、布尔数等，所有的文档内容都是字符数据。而 schema 则提供了丰富的数值类型，不但有整数、浮点数等常用的类型，还提供了自定义数据类型的机制。

一个 XML 文档只能使用一个 DTD 文档，尽管 DTD 可以通过多数实体机制来部分

改变这种缺点，但还是严重地导致了 DTD 的继承和使用的有限性。schema 则采用了名域空间的机制，使得一个 XML 文档可以调用多种 schema 文档。在代码的重用性和可扩展性方面要远远优于 DTD。

8.4.1 逻辑 XML Schema 的文档结构

XML Schema 则是一类特殊的 XML 文档，除了具有 XML 文档的语法要求外，还要有一些特殊的要求。下面给出了这种文档的一个模板程序 1，任何的 XML Schema 文档只要在该文档的基础上进行继续编写即可。

程序 1

文件 1.xml

```
<?xml  verslon="1.0" ?>
<Schema  xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">
</Schema>
```

所有的内容都添加在<Schema></Schema>当中，该程序可以作为所有 schema 文档的模板。关于 schema 的模板在语法方面总结几点：

（1）根标记必须为<Schema></Schema>。

（2）xmlns 是一个名域声明，表示 Schema 文档中使用的标记都是内标识为 urn:schemas-microsoft-com:xml-data 规范所定义的，前面字符串可以当作是该规范的名字。

（3）xmtns:dt 指定了文档中有关数据类型定义的标识都来自下面规范：

```
urn:schemas-microsoft-com:datatypes
```

上面的模板是一切 schema 文档都要遵守的一般性语法要求，下面我们将分别的从各个方面来介绍 XML Schema 的一些细节知识。

8.4.2 元素的定义

schema 中元素的定义使用标记<ElementType>来完成，具体的语法如下面的例子所示。

```
<ElementType name="name"/>
```

上面一条语句声明了一个<name>标记，该标记内容的类型是一般的文字，如果想指定为其他的数值类型，可以按以下的语法来制定。

```
<ElementType name="date" dt:type="date"/>
```

上面语句声明了一个<date>标记，它内容的数据类型是日期型的，其他数据类请参考表 8-5。

表 8-5　XML Schema 所提供的常用数据类型

数据类型名称	所表示的数据类型	数据类型名称	所表示的数据类型
Boolean	布尔型	Int	整数型
Char	字符型	Number	数字型
Date	日期型	String	字符串型
date time	日期时间型	URI	通用资源表示符
Float	浮点型		

此外还有 entity、entities、enumerationn、id、idref、idrefs、nmtoken、nmtokens、notation 等数据类型。

如果一个标记中含有子标记，则子标记的定义采用下面的语法：

```
<ElementType  name="books">
<elementt  type="book"/>
</ElementType>
<ElementType  name="book"/>
```

上面程序段定义了一个标记<books>，它有一个子标记。下面我们通过一个完整的程序 2 来说明标记的定义方法。

程序 2

文件 2.XML

```
<?xml  version="1.0"  ?>
<Schema  xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
<ElementType  name="books">
<element  type="book"/>
/ElementType>
<ElementType name="book"  dt:type="int"
</Schema>
```

下面是使用该 schema 的 XML 文档程序 3。

程序 3

文件 3.xml

```
<?xml  version="1.0"  ?>
<books  xmlns="x-schema:2.xml">
    <book>hello  world</book>
</books>
```

程序 3 中有一点需要注意：<books xmlns="x-schema:2.xml">该语句的作用是指定标

记<books></books>之间的所有标记都是在文件 2 中定义的。这也是 XML 调用一个 schema 文件的方法。

上面程序的错误在于，标记<book>的内容定义的是整数型，而实际的 XML 文档中使用的却是一个字符串，因而是一个不合法的文档。只要将<book></book>标记的内容改成整数型的就成为合法的文档，例如下面的程序：

```
<book>123</book>
```

1．元素内容类型的指定

有的标记只能自己独立使用，有的标记必须包含其他标记作为自己的子标记，决定一个标记语言的该方面的技术是元素的内容类型。这里可以通过关键字 content 指定这种属性。

Content 属性可以有下面的几种取值。

（1）textOnly：表示该标记只包含文本型的内容。

（2）eltOnly：表示该标记只包含元素类型的内容。

（3）empty：表示是一个空元素。

（4）mixed：表示上面的各种情况。

2．标记的子标记的出现次数

一个标记子标记的出现次数有 minOccurs、maxOccurs 两个属性来设置，它有如下的几种设置情况。

（1）子标记可以出现不止一次，即可以出现一次或多次，如：

```
maxOccurs="*"
```

如果一个标记的子标记可以出现一次或多次时，只要指定该标记的 maxOccurs 的属性值为“*”即可。

（2）只能出现一个，如：

```
maxOccurs="1"
```

在默认情况之下 maxOccurs 的属性值为 1，除非 content 的属性设置为 mixed，如果 content 的属性被设置成了 mixed，那么 maxOccuu 默认位为“*”。

（3）子标记可有可无，如：

```
maxOccurs="0"
```

如果一个标记的子标记指定了上面的属性，表示该子标记是可选的。

（4）子标记最少出现的次数，该次数通常是 0 或 1。如果 minOccurs 的值设置为 1 就表示该标记必须出现。

```
maxOccurs="1"
```

3．标记的子标记出现的次序

一个标记的子标记出现次序由一个属性 order 来控制，通常情况下该属性有三种取值。

（1）order="one"

如果一个标记设置了该属性，则表示此标记的子标记是它子标记列表中的任一个。

（2）order="many"

如果一个标记设置了该属性，则表示该标记的所有子标记可以按任意的顺序出现任意的次数。

（3）order="seq"

如果一个标记设置了该属性，则表示该标记的所有子标记必须按照定义好的次序出现。

4．元素的开放性和封闭性

元素的开放性和封闭性指的是元素能否包含定义以外子元素的问题，该方面的功能由属性 model 来描述，当该属件的值是 open 时，该元素是开放的，除了可以包含自己定义的子标记外，还可以包含其他的标记。当该属性的值为 close 时，元素除了包含自己定义时定义的子标记外，不能再包含别的标记。下面通过一个例子程序来演示该属性的应用。

```
12.xml
<?xml  version="1.0"?>
<Schema  xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">
<ElementType  name="books"  model="open">
<element  type="book"/>
<element  type="note/ >
<element  type="price"/>
<element  type="author"/>
</ ElementType>
<ElcmentType  name="book"  content="textOnly"/>
<ElemenLType  name="note"  content="textOnly"/>
<ElemenLType  name="price"  content="textOnly"/>
<ElemenLType  name="author"  content="textOnly"/>
</ Schema>
```

8.5 可扩展样式表语言

8.5.1 可扩展样式表语言概述

（eXtensible Stylesheet Language，XSL）是描述 XML 文档样式信息的一种语言，是

由 W3C 制定的。

前面已经讲过，XML 的一个优点就是形式与内容相分离，这使得 XML 文档具有简洁、易读的特点。它的样式信息都包含在称为样式单的样式文件中。而 XSL 就是它的两种样式单的其中之一。另一种是上一节介绍过的已经运用在 HTML 中的层叠样式单（CSS），是一种静态的样式描述格式，其本身不遵从 XML 的语法规范。而 XSL 不同，它本身就是一个 XML 文档，系统可以使用同一个 XML 分析器对 XML 文档及其相关的 XSL 文档进行分析处理。

XSL 是通过 XML 进行定义的，遵从 XML 语法规范，是 XML 的一种具体应用。它由两大部分组成：第一部分描述了如何将一个 XML 文档进行转换、转换为可浏览或可输出的格式；第二部分则定义了格式对象（Fomatted Object，FO）。在输出时，首先根据 XML 文档构造源树，然后根据给定的 XSL 将这个源树转换为可以显示的结果树，这个过程称为树转换，最后再按照 FO 分析结果树，产生一个可以在屏幕上、纸上、语音设备或其他媒体中输出的结果，这个过程称为格式化。

到目前为止，W3C 还未能出台一个得到多方认可的 FO，但是描述树转换的这一部分协议却日趋成熟，已从 XSL 中分离出来，另取名为 XSLT（XSL Transformations），其正式推荐标准于 1999 年 11 月 16 日推出，现在一般所说的 XSL 大多指的是 XSLT。与 XSLT一同推出的还有其配套标准 Xpath（XML Path Language，XML 路径语言），这个标准用来描述如何识别、选择、匹配 XML 文档中的各个构成元件，包括元素、属性和文字内容等。

使用 XSL 显示 XML 的基本思想是通过定义模板将 XML 源文档转换为带样式信息的可浏览文档。最终的可浏览文档可以是 HTML 格式、带 CSS 的 XML 格式及 FO 格式。

XSL 最初是由 Microsoft 公司提出来的，因此它对 XSL 的支持也比较好，Microsoft IE 5 已经部分支持 XSL。

在 XML 中使用如下语句声明 XSL 样式单：

```
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"?>
```

如前所述，XSLT 主要的功能就是转换，它将一个没有形式表现的 XML 内容文档作为一个源树，将其转换为一个有样式信息的结果树。在 XSLT 文档中定义了与 XML 文档中各个逻辑成分相匹配的模板，以及模板匹配和转换方式。值得一提的是，尽管制定 XSLT 规范的初衷只是利用它来进行 XML 文档与可格式化对象之间的转换，但它的巨大潜力却表现在它可以很好地描述从XML文档向任何一个其他格式的文档作转换的方法，例如转换为另一个逻辑结构的 XML 文档、HTML 文档、XHTML（The Extensible HyperText Markup Language）文档、VRML（Virtual Reality Modeling Language）文档甚至 SVG（Scalable Vector Graphics）文档。

XSL 在网络中的应用大体分为两种模式。

1．服务器端转换模式

在这种模式下，XML 文件下载到浏览器前先转换成 HTML，然后再将 HTML 文件送往客户端进行浏览。有两种方式：

（1）动态方式。即当服务器接到转换请求时再进行实时转换．这种方式无疑对服务器要求较高。

（2）批量方式。事先用 XSL 将一批 XML 文档转换为 HTML 文件，接到转换请求后直接调用转换好的 HTML 文件即可。

2．客户端转换模式

这种方式将 XML 和 XSL 文件都传送到客户端，需要浏览时由浏览器实时进行转换，前提是浏览器必须支持 XML+XSL 的工作方式。

本节将着重介绍 XSLT 对 XML 文档的显示转换功能，并将 XPath 作为 XSLT 的基础予以介绍。对于 FO，在此只用少量的篇幅介绍一下，使读者对其有一个概括性的了解。

8.5.2 XSLT 的常用句法和函数

1．文档结构

前面说过，XSLT 文档本身是 XML 文档，因此文档的第一句自然是：

```
<?xml version="1.0"?>
```

接下来是样式单部分：

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     ...
</xsl:stylesheet>
```

也可以写作：

```
<xsl:tranform version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Tranform">
     ...
</xsl:transform>
```

xsl:transform 与 xsl:stylesheet 具有相同的含义，都表示元素所包含的内容为样式单。xsl:stylesheet 元素必须包含有 version 属性，用以指示该 XSL 文档遵从哪一个版本的 XSL 标准。另外，xmlns:xsl 指示了 XSL 的命名空间，在 XSLT 标准中，定义了 XSLT 的命名空间为 http://www.w3.org/1999/XSL/Transform。

xsl:stylesheet 又可包含以下几种元素：

xsl:import　xsl:output　xsl:attribute-set　xsl:include　xsl:key
xsl:variable　xsl:strip-space　xsl:decimal-format　xsl:param　xsl:preserve-space
xsl:namcspace-alias　xsl:template

由此，一个样式单可以写成如下形式：

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:import href="..."/>
    <xsl:include href="..."/>
    <xsl:strip-space elements="..."/>
    <xsl:preserve-space elements="..."/>
    <xsl:output method="..."/>
    <xsl:key name="..." match="..." use="..."/>
    <xsl:decimal-format name="..."/>
    <xsl:namespace-alias stylesheet-prefix="..." result-prefix="..."/>
    <xsl:attribute-set name="...">
    ...
    </xsl:attribute-set>
    <xsl:variable name="...">...</xsl:variable>
    <xsl:param name="...">...</xsl:param>
    <xsl:template match="...">
    ...
    </xsl:template>
    <xsl:template name="...">
    ...
    <xsl:template>
</xsl:stylesheet>
```

通常把 xsl:stylesheet 的子元总称为顶级元素（top-1evel），顶级元素在 xsl:stylesheet 元素中出现的次序可以是任意的，没有固定的先后次序。每个样式单都包含有零个或多个顶级元素，有关各顶级元素的具体含义我们将在以后的小节中逐步介绍。

XSLT 在进行转换时，首先遍历 XML 源文档树，找到要处理的节点，然后将定义好的模板信息施加到该节点中。

2．模板与应用

xsl:template 是模板元素，通常每个 xsl:template 有一个节点匹配属性，由“match=”指定。在对模板进行匹配时使用 xsl:apply templates，选择要匹配的模板，相当于一个调用的过程。对节点的匹配规则遵照 XPath 的标准定义。

XSLT 对文档树的匹配总是从根节点“/”开始。<xsl:template match="/">就是匹配根

节点，然后把定义好的包含 HTML 文档的开始部分的模板施加给根节点。

在该例子中，定义了一个 roster 模板：

```
<xsl:template match="roster">
...
</xsl:template>
```

并在根节点的模板中调用了该模板：

```
<xsl:apply-templates seIect="roster"/>
```

不向的模板的设计，可以导致同一个文档有不同的输出效果的输出样式主要有以下两种方法。

1）定义不同的样式

例如，CAPTION（font-size:15pt;font-weight:bold:color:red）

2）利用 mode 属性

xsl:template 元素还有一个 mode 属性，利用这个属性可以根据需要去匹配不同模式的模板。

```
<xsl:template mdch="/" mode="blue">
    ...
<TITLE>学生花名册</TITLE>
<STYLE>
   CAPTION{font-size:15pt;font-weight:bold;color:blue}
</STYLE>
<TITLE>学生花名册</TITLE>
<STYLE>
   CAPTIONI{font-size:15pt;font-weight:bold;color:red}
</STYLE>
...
```

那么，如果要将表格标题输出为蓝色字，要用下面语句匹配：

```
<xsl:apply-templates select="/" mode="blue"/>
```

如果要将表格标题输出为红色字，则应写为：

```
<xsl:apply-templates select="/" mode="red"/>
```

以上介绍了利用模板达到不同输出效果的两种方法，第一种方法采取的是直接修改模板中内容，从而达到得到另外一种输出效果的目的。而第二种方法则是利用了 xsl:template 元素本身的 mode 属性，相对于第一种方法而言，具有更强的灵活性。

模板总是与节点相对应的，一个节点可能对应于不同的模板，那么在进行模板匹配时如何确定各模板匹配的先后次序呢？

在 XSLT 中，允许为 xsl:template 设置优先级，写法是：

```
<xsl:template match="student" priority="n">
...
</xsl:template>
```

n 为优先级数。

3．计算节点值

在使用 XSLT 进行转换时，常常需要获取节点值，使用 xsl:value-of 元素可达到这个目的，如：

```
<xsl:value-of select="origin"/>
```

得到的是学生原籍的值，select 属性指定要获取的是哪一个节点的节点值。

4．元素与属性创建

XSLT 是一个动态的样式单，在处理过程中可产生新的元素或元素属性。

1）创建元素

xsl:element 元素可在 XSLT 对文档转换时动态地生成新元素，如下例子中的 CAPTION 标记就可以用动态生成的方法来生成。

```
<xsl:element name="CAPTION">
学生花名册
</xsl:element>
```

XSLT 处理此句时，将生成下面的元素：

```
<CAPTION>学生花名册</CAPTION>
```

2）创建属性

xsl:attribute 元素可在 XSLT 对文档转换时动态地生成元素属性，如下面的例中 CAPTION 标记可以用动态生成的方法加入属性。

```
<CAPTION>
   <xsI:attribute name="style">
   color:blue
   </xsl:attribute>
   学生花名册
</CAPTION>
```

经 XSLT 转换后，将生成下面的元素：

```
<CAPTION style="color:blue">
```

```
    学生花名册
<CAPTION>
```

3）创建文本

xsl:text 元素可实现在样式单转换时动态创建文本，例如：

```
<xsl:template match="roster"/>
    <xsl:text>
    这是学生花名册
    </xsl:text>
</xsl:template>
```

在 XSLT 进行转换时，匹配到 roster 节点后，将输出文字：

```
这是学生花名册
```

从上面的例子中似乎看不出使用 xsl:text 的必要性，事实上 xsl:text 的优势在于它可以保护文本中的空白字符。对于处理特殊类型的文本是很有用处的。

4）创建处理指令

xsl:processing-instruction 元素可以实现在 XSLT 进行转换时自动生成处理指令，例如：

```
<xsl:processing-instruction name="xml-stylesheet>
    href="book.css"type="text/css"
</xsl:processing-instruction>
```

将创建如下处理指令：

```
<?xml-stylesheet href="book.css" type="text/css"?>
```

5）创建注释

xsl:comment 元素可在 XSLT 结果树中创建一个注释节点，例如：

```
<xsl:comment>
    以下是学生花名册，请勿删改!
</xsl: comment>
```

经 XSLT 转换后生成注释：

```
<!--以下是学生花名册，请勿删改-->
```

5．节点拷贝

在对 XML 文档进行处理时，XSLT 还可以通过拷贝的方式复制节点，方法是利用 xsl:copy 和 xsl:copy-of。

xsl:copy 只拷贝当前节点，不包括子节点和属性。而 xsl:copy-of 的拷贝内容则包括

当前节点、子节点和属性。

6. 循环处理

使用 xsl:for-each 可对所选节点依次进行处理，如在生成表格处理中利用循环将各个学生的信息取出放入表格中的，写法如下。

```
<xsl:for-each select="student" ord-by="name">
...
</xsl:for-each>
```

7. 排序

用 xsl:for-each 或 xsl:apply-templates 匹配的节点可使用 xsl:sort 将所选节点内容进行排序，如下所示。

1）按大小写排序

```
<xsl:sort case-order="upper-first" select="@id"/>
```

将以 id 为关键字按大写优先排序，而

```
<xsl:sort case-order="lower-first" select="@id"/>
```

将以 id 为关键字按小写优先排序。

2）按字母顺序排序

```
<xsl:sort order="ascending" select="@id"/>
```

将以 id 为关键字按字母升序排序，而

```
<xsl:sort order="desending"  select="@id"/>
```

将以 id 为关键字按字母降序排序。

3）按数据类型排序

在有文本和数字混合的内容排序时，可指定按哪种数据类型排序。

对于一组 id 数据 101，2，44，305 来说，如果写为

```
<xsl:sort data-type="text" select="@id"/>
```

排序结果是 101，2，305，44。而写成

```
<xsl:sort data-type="number" select="@id"/>
```

排序结果是 2，44，101，305。

另外，还有一种排序方式，就是在前面学生花名册例中所使用的 order-by：

```
<xsl:for-each select="student" order-by="name">
```

也可使得输出学生时按名字排序。

8．输出格式

XSLT 是一个转换语言，它的目的是将 XML 源文档转换为另一种格式文档，它的输出结果可以是 HTML 文档，也可以是带 css 的 XML 文档。具体的输出格式由 xsl:output 指定。如果要输出为 HTML 文档，则写为

```
<xsl:output method="html"/>
```

同样，要输出 XML 文档写为

```
<xsl:output method="xml"/>
```

另外 method "text"用来输出文本节点的内容，例如：

```
<xsl:output method="text"/>
<xsl:template match="A">
<xsl:text>&lt;!ELEMENT</xsl:text>
```

如果文档中不出现 xsl:output，将默认输出为 XML 文档，但如果在匹配模板时使用了<HTML>标记，则输出为 HTML 文档。输出为 HTML 文档时系统都会自动加上下面语句：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

8.6 其他相关规范

8.6.1 XPath

当 W3C 首次开始开发一种 XML 查询语言时，他们意识到 XPointer 和 XSLT 组正在尝试完成访问一个 XML 文档的特定段的功能。因此，Xpointer，XSLT 和 XPath 组进行了合作，指定为 XML 使用一种查询语言。XPath 给 XSLT 和 XPointer 提供一个共同、整合的定位语法，用来定位 XML 文档中各个部位。在 XSLT 和 XPointer 之间，使用一种通用的语法——XPath 来实现功能的共享。

XPath 采用简洁的、非 XML 的语法，它基于 XML 文档的逻辑结构，在该结构中进行导航。XPath 表达式通常出现在 URL 和 XML 属性值里。除了用于定位，XPath 自身还有一个子集能用于进行匹配，它能验证一个节点是否匹配某个模式。XPath 将 XML 文档描绘为树或节点的模型，节点的类型有根节点、元素节点、属性节点、文本节点、注释节点、名称空间节点和处理指令节点 7 种。

XPath 规范定义了两个主要部分：一部分是表达式语法，另一部分是一组名为 XPath

核心库的基本函数。

指向某个 XML 文档中一个特定节点的路径由三部分信息构成：一个轴类型、一个节点测试和谓词。轴类型有多种，用来指定所选节点和环境之间的关系。节点测试用来指定根据上下文节点要查找什么类型的节点，除元素和属性节点外，可能使用的其他节点测试包括通配符“*”、text()、node()、comment()、processing-instruction()等。谓词以“[”开始，以“]”结束。谓词可以通过使用内置的函数来过滤不需要的节点，从而到达所需节点。将轴、节点测试和谓词信息类型放到一起，会生成下面的位置步语法结构。

<轴>::<节点测试>[<谓词表达式>]

例如，child::sibling[position()=3]，这个 XPath 语句使用一个谓词来选择上下文节点的第 3 个 sibling 子节点。

位置步首先利用轴和节点测试计算，得到初始结果集，然后依次利用谓词进行过滤；初始结果集中满足所有谓词的就是返回结果。

位置步总结了 XPath 语言实际上是如何工作的。要到达一个特定节点的位置，必须使用一个 XPath 表达式或模式来列出到达那步骤，这个过程的每一步都被一个“/”字符分开，这些位置步和“/”字符组成位置路径。位置路径是 XPath 中最普通的表达式类型。

XPath 是用作 XSLT 和 XPointer 的对 XML 文档各部分进行定位的语言。但是，XPath 不是作为一种独立语言设计的，它只在与另一种语言（例如 XPointer 或 XSLT）一起使用时才有用。

8.6.2　XLink 和 XPointer

XLink 指定一个文档如何链接到另一个文档，XPointer 指定文档内部的位置，它们都是基于 XPath 推荐标准。现在对它们进行一下简要的介绍。

XLink 规范是 W3C 在 2000 年 2 月 21 日发布的工作草案。在 www.w3.org/TR/xlink 可以找到这个规范的最新版本。XLink 用于从一个文档链接到另一文档。下面是 W3C 在工作草案中描述的：

此规范定义了 3CML 链接语言（XML Linking Language，XLink），它允许为创建和描述资源间的链接而在 XML 文档中插入元素。它使用 XML 语法创建结构对象，能够描述现在的 HTML 简单的单向超链接，也能描述更复杂的链接。

这里有一个例子用来了解一下 XLink 大概是什么样子。不像 HTML 超链接，在 XML 中任何元素都可以是一个链接。用 xlink:type 属性来指定一个元素成为链接，在这里创建了一个简单的 Xlink。

```
<MOVIE_REVIEW xmlns:xlink="http://www.w3.org/1999/xlink"
     xlink:type="simple"
     xlink:show="new"
```

```
        xlink:href="http://www.starpowdermovies.com/reviews.xml">
        Mr.Blandings Builds His Dream House
    <MOVIE REVIEW>
```

在本例中，通过设置 xlink:type 属性为 simple 创建了一个简单的 XLink，很像 HTML 中的超链接。将 xlink:show 属性设置为 new，这意味着兼容 XLink 的软件应当在新的窗口或其他显示上下文中打开链接到的文档。另外，将 xlink:href 属性设置为新文档的 URI。URI 更为通用，并且链接不一定使用在这里使用的 URL 形式。

出于熟悉的缘故，以一个简单的 XLink 开始，因为它与 HTML 链接非常相似。除了基本的单向链接，也就是在这里创建的简单链接，也可以创建双向链接和多文档间的链接，甚至是文档集间的链接。另外，还可以做更多的事情，包括在被称为链接库的链接数据库中对的链接排序。

XLink 使能链接到一个指定的文档，但是经常需要更为精确的定位。XPointer 允许在文档内部定位，而不必改变文档嵌入特殊的标记。

在文档内部指定位置，XPointer 规范是基于 XPath 规范的。

它允许识别文档中指定结点，举例如下。

```
/child::*[position()=126]/child::*[position()=first()]
```

下面是 W 3C 关于 XPointer 的描述：

此规范定义了 XML 指针语言（XML Pointer Language，XPointer），为任何 URI 引用用作段标识符，URI 引用是用于定位 Internet 媒体类型 text/xml 或 application/xml 资源的。基于 XML 路径语言的 XPointer 支持在 XML 文档内部结构中寻址。它允许遍历文档树和选择其中的一部分，并且是基于各种特性的，例如元素类型、属性值、字符内容、相对位置等。

尽管 XPointer 基于 xPath 规范，但是 XPointer 规范在许多方面扩展了 XPath。

如何将 XPointer 加入文档的 URI 来识别文档中的指定位置?只需加上符号“#”，（遵从 HTML 对 URL 的用法，URL 指明链接的目标），然后是 xpointer()，并将想使用的表达式放入圆括号中。举例如下：

```
<MOVIE_REVIEW xmlns:xlink="http://www.w3.org/1999/xlink"
    Xlink:type = "simple"
    Xlink:show="new"
Xlink:href="http://www.starpowdermovies.com/reviews.xml#
     Xpointer(/child::*[position()=126]/child::*[position()=first()])"
     >
    Mr,Blandings Builds His Dream House
</MOVIE_REVIEW>
```

第 9 章　面向构件的软件设计

面向构件的软件设计是现在软件设计的重要方法，是软件生产线技术、软件工厂技术重要的理论基础。而在 20 世纪 90 年代之前，软件领域中的构件一直没有成功发展，这与软件领域中软件复杂度管理和软件复用粒度需求有关。术语“软件构件”的定义以及“构件化软件”等的仍然存在许多不同的观点，但构件概念已经在工程中被建立并且发展较为成熟，本章讨论软件构件的概念及构件化软件开发设计的方法与存在问题。

9.1　构件的概念

人们经常不做区分地使用术语“构件”和“对象”，以及像“构件对象”这样的短语组合。对象常被称为是类的实例，对象和构件都通过类或接口向外界提供服务。对于对象和构件之间的交互，设计人员常用模式描述，用框架规范。而构件和框架又都常被称为是白盒的或黑盒的。程序语言设计者还不断提出诸如名字空间、模块、包等各种名词。这种术语和概念泛滥的现象亟待改变，应该消除它们的冗余歧义，或者对它们进行阐明、解释与辨析。这些都让术语“构件”更加扑朔迷离。

9.1.1　术语与概念

构件技术蕴涵了太多概念。一个典型的例子就是含义众多的术语“对象”。随着时间的推移，模块、类和构件的概念都最终为“对象”所包括。最近，术语“软件构件”甚至到了把以前的普通对象也称作是构件的程度。将几个术语合并为一个，看起来似乎方便了使用，但除此之外再无其他好处。所以，必须在保证术语的准确度和直观性的前提下取得某种平衡。下面定义了几个关键的术语，并描述了它们之间的关系。

1．构件

构件的特性如下。

- 独立部署单元。
- 作为第三方的组装单元。
- 没有（外部的）可见状态。

这些特性有几重含义。一个构件是独立可部署的，意味着它必须能跟它所在的环境及其他构件完全分离。因此，构件必须封装自己的全部内部特征。并且，构件作为一个部署单元，具有原子性，是不可拆分的。也就是说，第三方没有权利访问其所使用的任何构件的内部细节信息。

在这样的约束下，如果第三方厂商能将一个构件和其他构件组装在一起，那么这个构件不但必须具备足够好的内聚性，还必须将自己的依赖条件和所提供的服务说明清楚。换句话说，构件必须封装它的实现，并且只通过良好定义的接口与外部环境进行交互。

最后，一个构件不能有任何（外部的）可见状态——这要求构件不能与自己的拷贝有所区别。但对于不影响构件功能的某些属性，例如，用于计费的序列号，则没有这种限制。通过对属性的可见性进行限制，允许用户在不影响构件的可见行为的前提下，使用合法的技术手段对那些影响性能的状态进行特殊处理。特别是，构件可以将某些状态专门用于缓存（缓存具有这样的特性：当它被清空时除了可能会降低性能以外，没有其他后果）。

构件在特定的系统中可以被装载和激活。但是，由于构件本质上没有状态，因此，在同一操作系统进程中装载多个构件的拷贝是毫无意义的，而且它们之间是不可区分的。也就是说，给定一个进程（或者其他的语境），至多会存在一个特定构件的拷贝。因此，谈论某个构件的可用拷贝的数量是没有什么意义的。

在目前许多系统中，构件被实现为大粒度的单元，系统中的构件只能有一个实例。例如，一个数据库服务器可以作为一个构件。如果这个服务器刚好只维护了一个数据库，那么会很容易把该数据库误认为是实例，如公司里的员工工资管理服务器。该数据库服务器连同其中的数据库，可以被视为一个有可见状态的模块。根据上面的定义，该数据库并不是一个构件，但那个静态的数据库服务器程序却是一个构件——它只支持一个数据库“对象”实例。也就是说，在这个例子中，工资管理服务器程序是一个构件，而其中的工资数据只是实例（对象）。这种将不易变的“模型”和易变的“实例”分离的做法避免了大量的维护问题。如果允许构件拥有可见状态的话，那么任何两个来自同一个构件的实例都不会拥有相同的属性。

在这一点上一定要分辨清楚。这里所说的构件的概念与对象层次上的可见或不可见状态无关，也与对象状态的生命周期（每次呼叫，每次会话，或是永久的）无关。这些全都是对象层次上所关心的东西，与构件的概念并没有直接的关系，但是通过构件，我们可以获得拥有任何这些属性的对象。

2．对象

说起对象，不得不提实例化、标志和封装。与构件的特性不同，对象的特性是：

（1）一个实例单元，具有唯一的标志。

（2）可能具有状态，此状态外部可见。

（3）封装了自己的状态和行为。

同样，对象的一系列属性随之而来。由于对象是一个实例化的单元，所以不能被局部初始化。由于对象有各自的状态，它必须有唯一的标志，以使它在整个生命周期内，无论状态如何变化，都能够被唯一地识别。

当对象被实例化的时候，需要一个构造方案来描述其状态空间、初始状态和新生对

象的行为。该方案在对象存在之前就已经存在。显式存在的实例化方案称为类。也有隐式的实例化方案，即通过克隆一个已存在的对象来实现，即原型对象。

无论使用类的形式，还是用原型对象的形式来初始化一个对象，这个新生的对象都必须被设置一个初始状态。创建与初始化控制对象的代码可以是一个静态的过程——如果它是对象所从属类的一部分，就被称为构造函数。如果这个对象是专门用来创建与初始化对象的，则简称为工厂。对象中专门用来返回其他新创建的对象的方法常被称为工厂方法。

3. 构件与对象

构件的行为显然可以通过对象来实现，因此构件通常包含了若干类或不可更改的原型对象。除此之外，构件还包括一系列对象，这些对象被用来获取默认的初始状态和其他的构件资源。

但构件并非一定要包含类元素，它甚至可以不包含类。实际上，构件可以拥有传统的过程体，甚至全局变量，它也可以通过函数语言，或者汇编语言，或者其他可用方法实现自身的全部特性。构件创建的对象——更确切地说是对这些对象的引用——可以与该构件分离开来，并对构件的客户可见。构件的客户通常是指其他的构件。除非构件的对象对客户可见，否则我们无从判断一个构件内部是否是“完全面向对象”的。

一个构件可以包含多个类元素，但是一个类元素只能属于一个构件。将一个类拆分进行部署通常没什么意义。另外，正如类之间可以通过继承关系等产生依赖一样，构件之间也可以存在互相依赖的关系——这种依赖很重要。一个类的父类并不一定与它的子类存在于同一个构件中。如果一个类的父类存在于外部的其他构件中，那么这两个类之间的继承关系便是跨构件的，这种关系将会导致相关联的构件间的导入关系。规约的继承是保证正确性的一项很关键的技术，因为共同的规约是构件间达成共识的基础。至于构件间对实现的继承是好是坏，仍然是众多学术流派争论的焦点。

4. 模块

构件与模块的概念其实非常类似。模块的概念出现于 20 世纪 70 年代后期的模块化语言（Wirth，1977；Mitchell 等人，1979）。最广泛使用的模块化语言是 Modula-2（Wirth，1982）和 Ada。在 Ada 里面，模块被称做包，但其实两者是相同的。模块化方法成熟的标志是其对分离编译技术的支持，包括跨模块的正确的类型检查能力。

随着 Eiffel 语言的面世，类被认为是更好的模块（Meyer，1988）。这似乎是正确的，因为我们最初的想法是每个模块实现一种抽象数据类型。毕竟，我们可以把一个类看成是一个抽象数据类型的实现，只不过它多了继承和多态的特性而已。然而，模块常常被用于把多个诸如抽象数据类型、类等实体打包到一个单元中。并且，模块没有实例化的概念，而类却有。

在其后出现的程序设计语言中——比如 Modula-3，Component Pascal，和 C#——模块的概念（C#中的集合）与类的概念是区分对待的。在任何情况下，模块都可以包含

多个类。在有些没有模块概念的语言（诸如 Java 语言）中，模块可以通过嵌套类来模拟实现。类之间的继承关系并不受模块界限的限制。另外值得一提的是，在 Smalltalk 系统中，经常会通过修改当前已存在的类来构造一个应用程序，人们已经开始尝试定义“模块系统”，这将使 Smalltalk 越过类而直接达到构件级水平，例如 Fresco (Wills，1991)。

模块本身就可以作为一个最简单的构件。即使不包含任何类元素的模块也可以实现构件的功能。传统的数值计算函数库就是一个很好的例子，这些库是功能性的，而不是面向对象的，但却可以打包成模块。然而，一个成熟的复杂的构件却并不是简单地仅用模块就可以实现的。模块没有持久不变的资源，有的只是那些被硬编码到程序中的常量。资源可以参数化一个构件。通过替换这些资源，就可以重新配置该构件而无需更改构件代码。例如，本地化设置可以通过资源配置实现。看起来资源配置好像为构件赋予了可变的状态值。但是我们知道，构件不能修改它们自身的资源，这些资源与编译后的代码一样只是构件的组成部分。追踪一个构件与它所派生的本地化了的构件之间的关联，在某种程度上，和追踪同一构件的不同的发布版本之间的关系相似。

某些情况下，模块并不适合作为构件，掌握这些情况是非常有用的。根据本书的定义，构件没有外部可见的状态，但是模块却可以显式地用全局变量来使其状态可见。并且，通过直接导入其他模块的接口，模块之间可以存在静态的代码依赖。而对于构件来说，虽然也允许存在对构件外部代码的静态依赖关系，但却并不提倡。这种静态依赖关系应被限定用于那些合约元素，包括类型和常量。使用间接而非直接的接口表示模块的依赖关系，把对实现代码的依赖关系限定于对象层次，就可以利用同一接口的不同实现来灵活地组装模块。

总的说来，模块化是构件技术产生的前提，但对于构件来说，传统的模块化的概念和标准是远远不够的。很多模块化的概念源自 Parnas（1972），其中包括最大化内聚性与最小化耦合性这条基本原理。因此，模块化的思想并不新鲜。但遗憾的是，现今的大部分的软件仍然不是模块化的。比如，有不少的大型企业应用都是对一个单一的数据库进行操作，允许应用系统的任何一部分依赖于数据模型的任何部分。但构件技术则要求系统中各部分必须互相独立，或者存在可控的显式依赖关系。因此构件技术必将导致模块化的解决方案。这种软件工程效益充分说明对构件技术的投资是有价值的。

5. 白盒抽象、黑盒抽象与重用

黑盒抽象与白盒抽象的区别主要在于接口“后面”的实现细节是否可见。在理想的黑盒抽象的情况下，客户对接口和规约之外的实现细节一无所知。而在白盒抽象中，在接口限制了用户行为并确保了封装性的情况下，客户仍然可以通过继承对构件的实现细节进行修改。由于在白盒方式中实现细节对外界是完全可见的，因此可以对实现细节进行研究，以加深对该接口抽象含义的理解。

揭示实现细节的可控部分。这是一个有争议的概念，因为部分可见的实现细节可以是规约的一部分。一个完整的实现只需要保证，能被客户看见的那部分实现细节与抽象

的接口规约一致即可。这是将规约实现的标准方式。

黑盒重用指仅仅依赖接口和规约来实现。比如，在绝大多数系统中，应用程序接口（Application Programming Interface，API）完全与内部的具体实现无关。用这样的应用程序接口构造系统相当于黑盒重用这些接口的实现。

相反，白盒重用指依赖于对具体实现细节的理解，通过接口来使用软件部件。大部分类库和框架都会提供源代码，应用程序开发人员通过学习类的具体实现，就可以知道如何构造该类的子类。

在白盒重用中，被重用的软件不可以轻易地被另外的软件替换。如果贸然替换将有可能破坏正在重用的客户端，因为这些客户端依赖于那些在未来可能发生改变的实现细节。

根据上述特性可以得出以下的定义："软件构件是一种组装单元，它具有规范的接口规约和显式的语境依赖。软件构件可以被独立地部署并由第三方任意地组装。"

这个定义最先是在 1996 年的面向对象程序设计欧洲会议上（European Conference on Object-Oriented Programming，ECOOP），由面向构件程序设计工作组（Szyperski 和 Pfister，1997）提出。该定义涵盖了我们之前讨论的那些构件特性。它既包括了技术因素，例如独立性、合约接口、组装，也包括了市场因素，例如第三方和部署。就技术和市场两方面的因素融为一体而言，即使是超出软件范围来评价，构件也是独一无二的。

而从当前的角度看，上述定义仍然需要进一步澄清。一个可部署构件的合约内容远不只接口和依赖，它还要规定构件应该如何部署、一旦被部署（和启动）了应该如何被实例化、实例如何通过规定的接口工作等。事实上，各个接口的规约都应该被独立地看待，任何提供与使用该接口实现的构件之间都是相对独立的。比如，一个实现队列操作的构件通过一个接口获得物理存储空间，通过另外两个接口提供入队列和出队列的操作。在构件的合约中说明，通过入队列接口插入队列的元素，可以通过出队列接口中的操作取出来，这种关联关系，任何接口规约都不能单独提供。该合约同时也规定构件一旦被实例化，就必须在关联一个实现了物理存储空间接口的构件之后才能被使用。这种关联将受到底层构件模型的组装规则的影响。具体的部署和安装的细节由特定的构件平台提供。

6．接口

接口是一个已命名的一组操作的集合。构件的客户（通常是其他构件）通过这些访问点来使用构件提供的服务。通常来说，构件在不同的访问点有多个不同的接口。每一个访问点会提供不同的服务，以迎合不同的客户需求。强调构件接口规范的合约性非常重要，因为构件和它的客户是在互不知情的情况下分别独立开发的，是合约提供了保证两者成功交互的公共中间层。

成功的合约接口需要遵循哪些非技术因素？首先，必须时刻关注经济效益。一个构件可以有多个接口，每一个接口提供一种服务。有一些服务会格外受客户欢迎，但是如

果所有服务都不受欢迎，那么服务的组合也不会受欢迎，这个构件就没有市场价值了。这样的话，就没有必要在非构件的实现方案的构件化上进行投资了。

其次，应当避免不当的市场分化，因为这威胁到构件的生存。所以，尽量不要重复引入功能相近的接口。在市场经济中，这通常是主要生产商在市场早期努力推行标准化的结果，或者是经过残酷的市场竞争优胜劣汰的结果。但是，前者可能会由于笨拙官僚的“委员会设计”问题而不能达到最优；而对于后者，市场竞争的非技术本质也可能导致结果不是最优的。

最后，为了使一个接口的规范和实现该接口的构件得到广泛应用，需要有一个公共传媒来向大众进行宣传和推广。要做到这一点，至少需要几种能被广泛认可的保证命名唯一性的命名方案。接口标准化的一个非常有意思的变种，是对消息的格式、模式和协议的标准化。它不是要将接口格式化为参数化操作的集合，而是关注输入输出的消息的标准化，它强调当机器在网络中互连时，标准的消息模式、格式、协议的重要性。这也是因特网（IP、UDP、TCP、SNMP 等）和 Web（HTTP、HTML 等）标准的主要做法。为了获得更广泛的语义，有必要在一个单一通用的消息格式语境中标准化消息模式。这就是 XML 的思想。XML 提供了一种统一的数据格式。

7. 显式语境依赖

在上文的构件定义中，构件除了要说明所提供的接口外，还要说明其需求。也就是说为了使构件正常地工作，必须说明其对部署环境的具体要求。这些要求被称为语境依赖，指的是构件组装和部署的语境，包括了定义组装规则的构件模型和定义构件部署、安装和激活规则的构件平台。如果只存在一种软件构件体系的话，那么只需要列举该构件所需的所有其他构件提供的接口，这样就足够可以说明全部的语境依赖。例如，一个合并邮件的构件会声明它需要一个文件系统的接口。但是，今天的大多数构件即使连这样的需要的接口也通常不进行声明。而构件提供的接口更受关注。

事实上，目前有几种构件体系同时存在，它们相互竞争，彼此冲突。例如，现在就有 OMG 的 CORBA，Sun 的 Java，以及微软的 COM 和 CLR（Common Language Runtime）等体系。并且，因为要支持不同的计算和网络平台，构件体系本身并不是单一的。这种状况还没有很快改善的迹象。而另外一种观点则认为，所有这些构件体系最终都只能归结为两类——CORBA+Java 体系和微软体系（包括 COM+和.NET/CLR）。但即使构件体系被刻意减少到不能再少的区区两种，在具体实现层次上还是存在着千差万别的。

8. 构件的规模

显然，构件只有在提供了“恰当”的接口集，以及对语境依赖没有严格限制的情况下，该构件可以在所有的构件体系中运行，并且其依赖的接口不会超出那些构件体系所能提供的范围时最好用。然而，只有极少的构件拥有这么弱的环境依赖性。技术上来说，一个构件可能和它所需要的所有软件捆绑起来被提供，但这显然违背了使用构件的初衷。注意，环境需求往往取决于构件运行的目标机器。如果是虚拟机，例如 Java 虚拟机，这

就显然是该构件体系规范的内容之一。如果是本地代码平台，仍然有类似于 Apple 的将多个二进制文件打包成一个文件的 Fat Binaries 这样的机制，可以使构件在“所有地方”运行。

构件设计者通常不会构造自给自足的构件，将所需的所有东西都打包进来，而是采取一种“最大化重用”的策略。为了避免在构件中重复实现那些次要的服务，设计师通常会只实现该构件的核心功能，然后重用其他所有的一切。面向对象的设计有向这种极端发展的趋势，许多面向对象的方法论者都大力提倡这种最大化重用的思想。

虽然最大化重用的思想有很多为人称道的优点，但是它也有一个潜在的缺点——语境依赖的爆炸性增长。如果构件在发布后其设计一直冻结不变，同时所有的部署环境也都一样，那么这个问题就不会出现。然而，构件会不断地演化，不同的部署环境会提供不同的配置，多种版本会同时存在，在这样的情况下大量的语境依赖只会使构件成为众矢之的。语境依赖越多，能满足构件环境需求的客户构件就越少。总之，最大化重用降低了可用性。

构件设计者需要为以上两者找到一个平衡点。当要描述构件的基本接口的时候，设计者们就需要做出抉择。增加语境依赖通常会使构件因重用而简洁，但却会降低可用性。此外，还必须考虑环境的演化会使构件更加脆弱，例如引入新版本带来的变化。增加构件的自给性可以减少语境依赖，增加可用性，并且使构件更健壮，但却会使构件规模过大。

9.1.2　标准化与规范化

通过提高接口与构件体系的标准化和规范化程度，可以使上文优化问题中的最优点偏向于简洁性一侧。一个东西越稳定，越容易被广泛接受，其成为某个构件的特殊需求的危险就越小。如果语境依赖能够被广泛支持，就不是什么缺点。比如仅仅在 50 年前，要求客户必须拥有电话才能谈妥生意是极不明智的。而现在，在世界上的许多地方，拥有电话已经成为必要条件。

1．通用市场与专业市场

如果要制定一种覆盖所有领域、有广泛市场的标准，就有必要区分面向通用市场与面向专业市场的两种标准。通用市场覆盖了多数甚至全部不同的市场领域；它对所有或绝大多数的客户和生产商都有影响。专业市场往往只限于某个特定的领域，相对来说影响比较小。例如，因特网和万维网的标准都属于通用市场标准。与之相反，放射医学领域的标准就只影响一个比较窄的专业市场，却同样会占有相当大的市场份额。

通用市场的标准化是非常困难的。如果有一项服务几乎和每个人都相关，那么它就得满足所有人的需求。想像一下那些通用程序设计语言标准化委员会，他们为顾全各方面的利益而疲于奔命。与此同时，成功的标准只有在通用市场中才能形成最广泛的影响，网络标准就是其中最好的例子。

令人吃惊的是，专业市场的标准化与通用市场同样艰辛，虽然原因各不相同。由于专业市场中涉及的人相对较少，所以比较容易形成一种折中的方案。然而，如果某个专业领域正在考虑标准化，那么为了培育市场，该领域就不能够太窄。由于所涉及的人较少，市场经济的机制就不容易很好地发挥作用，也就不太可能在短期内找到理想的、成本效益好的解决方案。

2．标准的构件体系与规范化

在基本构件体系和那些最重要的接口合约形成标准，并且在这些标准被相关的工业界支持的情况下，构件技术最为成功。然而，要发挥标准化的作用，就必须使与之竞争的其他标准的数目尽量很小。如果某个标准背后有一个强大的国际标准化组织认可，有一个非常有实力的企业推动，有众多有影响力的公司或组织联合支持，那么一切自然不成问题。然而，通常却是几个标准在相互竞争。如果因专业市场各自为政，导致某标准在不同领域重复建设，而该标准又可能适合其他领域，就可能出现戏剧性的结果：很多原先互不知情的标准竞争者在一夜之间一起出现。例如，放射医学和射电天文学就可以共享多种图像处理标准。

如果相互竞争的标准过多，而其相应的市场份额过小，就可能引发危机，这个问题可以通过规范化的手段来解决。公布共同的设计“模式”，并对其进行编目，原来互不知情的各方标准化实体就有可能在各自的目标领域发现共同点。当然，寻找和利用共同点的努力是否值得，即成本效益是否理想是个规模效益的问题。

9.2 构件的布线标准

“布线”用于连接电子构件。无论天然气、水或排污系统，对于所有要连接的部件来说，管道工程本质上是相同的。对于可连接的构件来说，这一级的标准是重要的。但是，需要注意的是不要高估了“布线”标准的重要性。比如说，庞大的世界范围内的电话系统之间的互联互通就是一个例子。

9.2.1 布线标准从何而来

由于过程的交互为进程边界所限，所以操作系统支持多种多样的进程间通信（Internet Process Connection，IPC）机制，典型的例子有文件、套接字（socket）及共享内存。除了 BSD-UNIX 套接字外，这些机制都不能跨平台移植。

IPC 机制的一个共同的优势是：它们可以很容易地被扩展到网络甚至是因特网上。这是传统进程模型的直接结果。在这个传统进程模型中，每个进程产生了一个幻象，就好像一个共享的物理主机上的每个进程都拥有单独的虚拟机。

RPC 的设想是在本地被调用者和远程调用者两端都使用指代（stub）。调用者使用严格的本地调用约定，就像调用了一个本地被调用者，实际上，却调用了一个本地指代

来编排（串行化）参数，并把它们发送到远端。在远端，另一个指代接收参数，并还原（反串行化）参数，然后调用真正的被调用者。和调用者一样，被调用者的过程本身也要遵循本地调用约定，并且不知道自己被远程调用了。编排和还原过程负责转化数据值，将它们从本地表示转化为网络格式，然后再转化为远端表示。通过这种方法，格式的差异等被跨越了。

分布式计算环境（DCE）是 OSF（Open Software Foundation，Open Group 的一部分）的一个标准，它是在跨越异构的平台上实现 RPC 机制的最重要的服务。在另一个极端，轻量 RPC 变化能被用来处理单机上的 IPC 问题。例如，Windows 支持跨进程的轻量 RPC；DCOM 出现后，可以支持不同机器间的完全 RPC。DCE 也通过对每个服务附加主版本号来支持版本控制。客户可以指定他们想要版本的服务。

潜在的透明性既是 RPC 的优点，同时也是其负担。因为它隐藏了本地调用、进程间调用及机器间调用的很重要的代价上的差异。在大多数当前的体系结构上，进程间调用比本地调用慢 10～1000 倍，而机器间调用比进程间调用慢 10～10 000 倍。

接口定义语言（Interface Definition Language，IDL），保证了不同环境下过程调用语义的一致性。对每一个可以被远程调用的过程，IDL 指定了参数的数目、传递模式和类型，以及可能的返回值的类型。为了确保跨机器边界的通信正常工作，所有的 IDL 都必须固定基本类型的范围，例如指定整数是 32 位的二进制的补码值（二进制的补码表示是一种以二进制形式表示负数的数学方案）。

过程调用及它们的二进制调用约定提供了一个良好证明的“布线”标准。但是它们还不能直接支持对象所需要的远程方法调用。如果和动态链接库（Dynamic Link Library，DLL）结合起来，远程过程调用就会在为构件“布线”形成一个有用的基础的过程中更进一步。服务可以通过名字（DLL 的名字）来定位，并被动态地绑定（不仅在编译时刻），而且服务可以是远程的。今天，Web 服务似乎已成为事实上的构件布线标准。

9.2.2　从过程到对象

使对象调用与过程调用区别开来的首要因素在于它们很靠后的、数据驱动的调用代码选择。一个方法调用，除非是优化过的，否则总会去检查接收消息的对象的类，并从该类提供的方法中选择方法实现。而且，一个方法总是将目标对象的引用作为另一个参数向外提供，该引用是消息发往的地址。面向对象编程的大多数优点来自方法调用的属性。

有趣的是，当前的对象调用并不遵循标准平台调用约定。原因很简单，就是由于当前的操作系统和它们的库有过程化的接口，因而对操作系统提供商来说从来就没有必要定义方法调用约定。结果，使用不同的编译器编译出来的代码就无法互操作，甚至使用同一种语言实现的代码也不行。为了通用，一个面向对象的库必须以源代码的形式发布。这也是为什么可执行类库（而不是源代码）在这种面向对象的情况下远不及在过程化的

情况下流行的原因。

在实现了过程调用的机器上实现方法调用是可能的。比如 IBM 的系统对象模型（System Object Model，SOM）就是这么做的。在 SOM 中，所有的语言绑定只是简单地调用 SOM 库过程，然后在 SOM 运行时刻动态地选择要调用的方法。CORBA 的 ORB（Object Request Broker）是另一个例子——最新版本的 SOM 实际上就是基于 CORBA 的。Microsoft 的 COM 也非常接近于仅使用过程调用约定，虽然它依赖包含函数指针的过程变量表（也称分配表）。但这不是问题，因为过程变量很久以来已经是调用约定的一部分。

另一个可能的方法是为方法调用定义一个带有内建支持的新的虚拟机层。这就是 Java 虚拟机和.NET 的运行时刻公共语言采用的方法。但是，和库的支持及系统范围的调用约定不同，虚拟机可能阻止或干扰超越它的边界的互操作。因此，JVM 和 CLR 都为跨越虚拟机边界的互操作提供特殊的支持。

9.2.3 深层次问题

如果在执行层上过程调用约定几乎是能胜任的，那为什么还要有这么多不同的竞争的提议呢？原因在于，为了实现互操作，还有其他的重要方面需要被考虑和标准化。需要回答的问题包括“接口如何指定”、“当离开它们的本地进程后对象引用如何处理”、“服务如何被定位和提供”及“构件演化如何处理”。

1．接口和对象引用规范

什么是接口？所有当前的方法一律将接口定义为一个已命名的操作的集合，每个操作带有一个已定义的特征标记（signatrue）和可能的返回值的类型。操作的特征标记定义了该操作的参数的数目、类型及传递模式。接口和什么相连接？对于这个问题，每种方法的处理是不同的。那些基于传统的对象模式的方法在接口和对象之间定义一个一对一的关系（CORBA 2.0，SOM）。对象在接口后面提供状态和实现。其他的方法将多个接口和一个单独的对象联系起来（Java，CLR），或者把多个接口和一个构件对象的多个部分对象联系起来（CORBA 3.0 的 COM，CCM）。明显地，一旦出现一个接口后面有多个对象的情况，就有身份标志的问题产生并且需要处理（为了这个原因，COM 和 CCM 提供了一个特殊的接口）。

如何指定接口？所有传统的处理方式都遵循 DCE（Data Communications Equipment），并且使用 IDL。遗憾的是，真正使用的不是单个的 IDL，而是存在着的几个竞争的提议，特别是 OMGIDL 和 COMIDL 这两个最强大的竞争者。Java 和 CLR 没有 IDL，这里，相关信息作为元数据被保留并可以映射到任何被支持的语言。程序员可以使用他们熟悉的语言来看待接口和其他类型定义，而不需要去学习他们熟悉的语言之外的一种 IDL。所谓的 Java IDL，实际上是一种结合了一个 IDL 到 Java 编译器的 Java 可调用的 CORBA ORB。通过一个从 Java 类型到 OMG IDL（反之亦可）的映射来支持 Java。在 OMG IDL 和 COM IDL 之间也已经定义了一个相似的双向映射。目前 OMG IDL 和

CLR 类型之间或者和更多的特定于 C#的类型之间还没有相似的映射。

什么是对象引用？当它们作为一个远程方法调用的参数被传递时是如何被处理的？每种方法的做法是不同的。但所有的方法都有一些机制，用于把本地的有意义的引用映射到包含跨进程、机器及网络边界的引用。

2．接口关系和多态性

所有的方法都规定了多态性。在所有的情况下，某个具有一个已知接口的实体可以是多个不同的、可能实现中的一个。同时，在所有的情况下，一个实现所能提供的方法比接口指定的要多。

在细节上，所有的做法都是不同的。CORBA 2.0 遵循一个传统的对象模型。一个对象有一个单独的接口，但这个接口可能是由其他的接口使用接口多继承组合而成的。实际提供的接口可能是所期望接口的子类型，其额外附加的能力可以被动态地发现。CORBA 3.0 中的 CCM 支持多接口继承及在它们间动态的导航。COM 也明确地支持除了被共同支持的已经提供的接口之外的必需的接口。COM 拥有不变的接口，也就是说一旦发布则不可扩展或修改。单接口继承被支持用来从已发布的接口派生出新的接口。但是，一个 COM 对象可以拥有多个接口。对一个特定的对象来说，它提供的接口的集合可能会随着时间的流逝而变化，并且可以被动态地发现。通过约定，COM 支持必需的接口。一个 Java 对象也可以实现多个接口，但是这更接近于多接口继承而不是 COM 那样的完全分离的接口，一个对象所实现的接口的集合由这个对象的类静态地确定。同样的情况也适用于 CLR 对象。Java 接口的多继承的传统在试图支持有冲突的方法名的接口时会导致问题的出现——Java 对象不能支持多个这样的接口。COM 和 CLR 都维护不同接口上的方法在实现层次上的分离，而不关心方法的名字。CLR 模型在支持多接口继承方面也超出了 COM，像 Java 或 CORBA。和 COM 一样，Java 和 CLR 仅通过约定的方式采取了支持必需的接口的做法。

3．命名和定位服务

接口是如何命名的？它们是如何相互关联的？没有两种做法对这两个问题的处理是完全一致的。COM 采用了 DCE 的 UUID（Universally Unique Identifier）的做法，在 COM 中被称为全局唯一标志符（Globals Unique Identifiers，GUID）。GUID 用来唯一地命名多样性的实体，包括接口（Image Impoet Descriptor，IID）、接口的组（称为分类（Category ID，CATID））以及类（Class ID，CLSID）。OMG CORBA 最初是把唯一命名留下用来单独实现的，依赖语言的绑定来维护程序的可移植性。在 CORBA 2.0 中，引入了全局唯一标志符。这些既可以是 DCE UUID，也可以是类似于常见的用于万维网的统一资源定位符（URL）的字符串。Java 完全依赖由内嵌的命名包来建立的唯一的名字路径。CLR 提供类似的有资格的名字来建立一个可读的命名方案，但最后把所有的名字都放入那些所谓的强组装名字（strong names of assemblies）。其实有很大的可能性，私有/公有密钥对中公有的那一半是独一无二的。这就是说，尽管标志符使用唯一的标志符来

支持个别的名字，但 CLR 可以使用一个独一无二的标志符来支持整个名字家族，只要这样的名字家族是在一个单独的组装中（一个 CLR 软件构件）被一起发布的。通过给定一个名字，所有的服务都提供一些注册表或者库的分类来帮助定位相应的服务。在这个类似目录的功能之上，所有的方法都还提供某种程度上的关于可用服务的一些元信息。所有方法都支持的最小的功能是，对被提供的接口类型的运行时刻的测试、接口的运行时刻反射及新实例的动态生成。

4．复合文档

软件构件的第一个实用方法是复合文档模型。复合文档是一个模型，对那些构成，即用户来说，在模型中构件及其合成具有直观的意义。Xerox Star 系统是第一个基于 Xerox Palo Alto 研究中心的研究结果的，但是没有能够获得足够的市场和观念份额。第一个突破是苹果公司的 Hypercard，因为它有简单直观的合成及使用模型，但是创建新的构件却是一件困难的事情。Microsoft 的 Visual Basic 也遵循了它，对 Visual Basic 控件有一个合理的编程模型。有了 Microsoft 的 OLE 和苹果公司的 OpenDoc 技术后，一般的文档都可以遵循它了。后来，嵌入对象比如 Java applets 和 ActiveX 控件的网页的出现，增加了一个新的维度。

在 OLE 中，复合文档的概念更前进了一步。首先，任意的容器可以被允许。除 Visual Basic 表单外，Word 文本、Excel 电子表格、PowerPoint 幻灯片等，都变成了 OLE（Object Linking and Embedding）容器。而且，“控件”的概念被推广到任意的文档服务器中去。然而最大的变化是构件可以同时是文档容器和服务器。结果，Word 文本能被用来注释一张 PowerPoint 幻灯片，而这张幻灯片也能被嵌入到另一 Word 文本里去。

复合文档还可以是一个把对象嵌入到了 HTML 页面里的网页。浏览器为所有的 Web 页面提供一个统一的文档模型。嵌入对象，比如 Java applets，能根据需要加入细节。但是，现在有了强大的服务器端 Web 编程模型，例如 Sun 的 JSP（Java Server Page）和 Microsoft 的 ASP（Active Server Pages）及目前的 ASP.NET。当合成一个基于后端数据和用户输入相结合的复合文档的时候，现代 Web 页面远远超过 OLE 技术的地方，正是这种服务器端模型。

9.2.4 XML

尽管 1998 年才出现，但 XML（扩展标记语言；W3C，2000b）在此前大量尝试都失败的地方获得了成功。一部分原因是由于 XML 的一些有趣的属性；一部分是由于合适的时机。XML 的到来与一个数量大量增长着的领域相关，特别是电子商务的领域需要用 XML 标准化。

XML 对于表示任何（半）结构化的数据十分有用，新的 XML 的应用就应运而生了。除了消息、Web 页和传统文档之外，现在普遍用 XML 来配置数据，即使这些数据从未被其他任何应用（而不是为之定义方案的应用）处理过或者阅读过，使用 XML 仍

然是有用的。浏览器比如 Internet Explorer，直接支持显示和查找 XML 文档。还有很多的工具为基于 XML 的数据提供其他普遍支持的形式，这些工具包括编辑器、方案检查器和方案驱动的翻译器。没有异常情况的话，XML 在独立起源和操作的应用间作为一门公共语言来使用会获得最大的好处。在某种意义上，XML 成了从协议和“布线”格式层到持久数据表示层的“布线”标准的概念。

9.3　构件框架

9.3.1　体系结构

系统的体系结构是任何大规模软件技术的关键基础，在基于构件的系统中起着至关重要的作用。只有当整体的体系结构良好地定义和维护，构件及系统的升级和维护才会有坚实的基础。构件体系结构的核心包括：构件和外部环境的交互；构件的角色；标准化工具的界面；对最终用户和部署人员的用户界面等。

1．体系结构的角色

体系结构是关于一个系统的整体视图，一个体系结构从总体上定义了总体的不变性，即那些根据这个特定的体系结构建立起来的所有系统的共同属性。体系结构把核心资源分类，以支持在资源竞争下的独立性。操作系统就是一个很好的例子，通过定义独立的进程之间如何竞争资源，操作系统部分地定义了运行于其上的系统所采用的体系结构。

体系结构为所有涉及的机制规定了恰当的框架，限制自由度，以控制变化性并支持协作。体系结构包括了所有支持独立使用机制进行互操作的策略决策。策略决策包括构件的角色。

体系结构需要基于对整体功能、性能、可靠性和安全性的主要考虑过细的决策可以放在一边，但关于所期望层次功能和性能的指导是必须的。例如，体系结构可能确切地规定一些细节来保证性能、可靠性或者安全性。在安全关键的应用中，有强调这些所谓非功能需求的传统。在任何体系结构中把这 4 个方面都作为一个整体的高优先级问题仍然是一个重要的目标。

2．概念化

在概念层次上，划分层次、标志构件、分离关注点的作用是显而易见的。但在一个具体的体系结构中，它们是否还存在？更具有争论性的是，超越对象的粒度是否真有必要？有趣的是，有时认为对象最主要的优势是对象和对象间的关系在需求、分析、设计和实现等阶段是一致的（Goldberg 和 Rubin，1995）。这种说法的成立需要两个前提，一是如果在所有的上述过程中都只有对象起主要作用；二是系统中所有超越对象的事物都可以被隔离。这两个前提也是所谓的“纯”面向对象方法的主要动机。

显然，并不是所有的事物都是对象。然而，任何需要一组对象进行交互的系统都可以通过指定一个代表对象来抽象这个交互对象组。此时，区分“has　a”（或者“contain a”）联系和“use　a”关系就变得很必要了。这个代表对象“包含”（“has　a”）对象组，而组中的对象之间也可以通过代表对象的协调而相互使用（use）。以图的形式建模对象之间的关系时，对象是节点，联系是这些节点间的有向边。“has　a”和“use　a”分别是图际边和图内边。让我们考虑在时间和空间语境中支持对象转换的外部服务，例如，存储复合文档。在典型的外部行为中，图内边需要追溯下去；图际边不需要追溯，但需要抽象地保持为“连接”，连接象征性地代表了有向边的目标节点。

3．构件系统架构特性

- 构件系统体系结构由一组平台决策、一组构件框架和构件框架之间的互操作设计组成。

平台是允许在其上安装构件和构件框架的一个基础设施，支持构件和构件框架的实例化和激活。平台可以是实际平台，也可以是虚拟平台。实际平台提供了直接的物理支持——也就是在硬件上实现了它们的服务。虚拟平台（也可以称做平台抽象或者平台外壳）在其他平台之上仿真了一个平台，以支持灵活的成本权衡能力。

- 构件框架是一种专用的体系结构（通常围绕一些关键的机制），同时，也是一组固定地作用于构件层次机制的策略。

构件框架常常实现一些协议以连接构件，并强制实施一些由框架决定的策略。管理如何使用框架自身所用机制的策略并不确定。实际上，它们可以留给更高一层的体系结构来确定。

- 概念框架的互操作设计包括系统体系结构连接的所有框架间的互操作的规则。

这样的设计是第二等的构件框架，构件框架可以看成是它的内插构件。到现在我们可以确信第二层次是必要的——包含所有内容的单个构件框架是不切实际的。现在还不清楚第三层或者更高的层次是否必要，但此处暗示的元体系结构模型是可扩展的，允许增长。

- 构件是一组通常需要同时部署的原子构件。构件和原子构件之间的区别在于，大多数原子构件永远都不会被单独部署，尽管它们可以被单独部署。

相反，大多数原子构件都属于一个构件家族，一次部署往往涉及整个家族。

- 一个原子构件是一个模块和一组资源。

原子构件是部署、版本控制和替换的基本单位。原子构件通常成组地部署，但是它也能够被单独部署。一个模块是不带单独资源的原子构件（在这个严格定义下，Java 包不是模块——在 Java 中部署的原子单元是类文件。一个单独的包被编译成多个单独的类文件——每个公共类都有一个）。

- 模块是一组类和可能的非面向对象的结构体，比如过程或者函数。

显然，一个模块可能静态地需要另一个模块的存在才能起作用。因此，一个模块只

有在其依赖的所有模块都已经可用后才能部署。这个依赖图必须是无循环的，否则一组循环依赖关系的模块总是需要同时部署，这就破坏了模块定义的性质。

- 资源是一个类型化的项的固定集合。

资源这个概念可以包含代码资源，进而包含模块。问题在于除了编译器编译一个模块或包生成的资源外，还可能存在其他的资源。在“纯对象”的方法中，资源是外部化的不可改变的对象——不可改变是因为构件没有持久化的标志，而且复制不能被区分。

4. 分层的构件体系结构

层的概念和层次分解在构件系统中十分有用。构件系统的每一个部分，包括构件本身，都可以被分层，因为在一个更大的体系结构中，构件可以被定位到特定层次。为了控制更大型的构件系统的复杂性，体系结构自身也需要分层。

如前所述的构件系统体系结构具有一组开放的构件框架。这组构件框架形成了第二水平层次，而每个构件框架都定义了第一水平层次的体系结构。在这里，区分水平分层和传统的垂直分层之间的本质区别非常重要。传统的垂直分层，自底向上地，抽象程度渐增，与应用相关的性质逐渐提高。在一个良好的垂直分层系统中，各个层次都应该考虑相应的性能和资源。相反，水平分层是性能和资源相关性递减而结构相关性渐增的。不同的水平层次关注不同的集成性，但都与同一个应用相关。图 9-1 描述了在一个三水平分层多垂直分层的体系结构中垂直分层和水平分层的相互影响。如同描绘的那样，高水平分层提供了共享的低垂直分层以集成低水平分层。水平分层被描绘成相邻的，而垂直分层则是一个叠于另一个的上面。

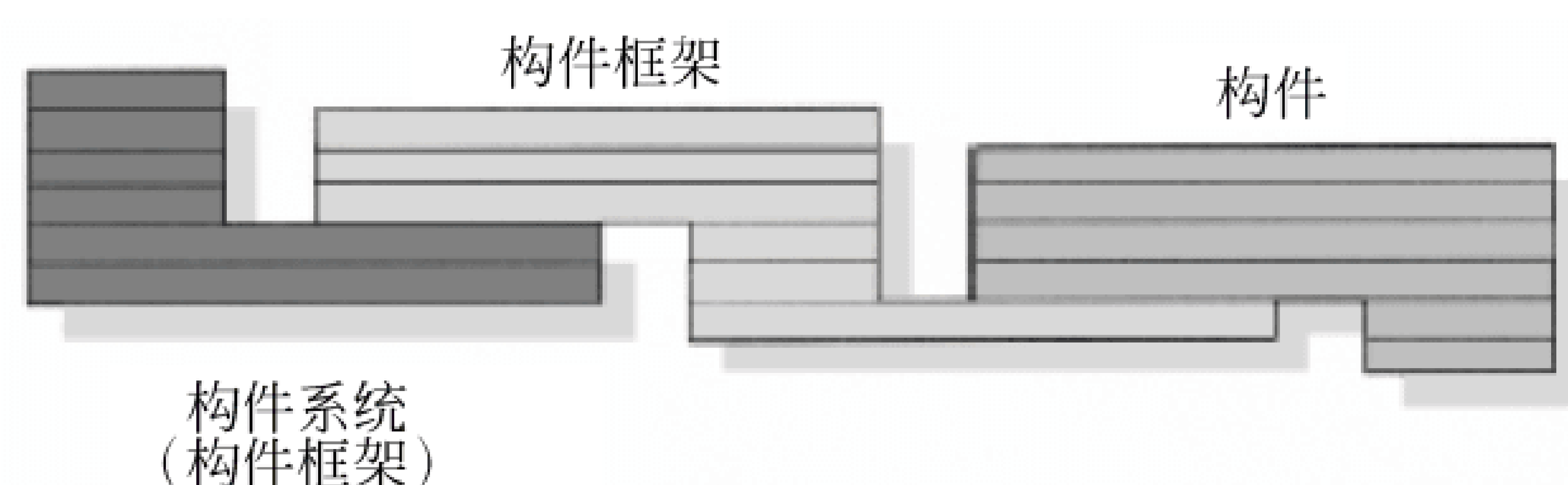


图 9-1　三水平分层多垂直分层的体系结构——构件、构件框架和构件系统

图 9-2 展示了构件实例之间如何相互通信。它们可以直接通信（例如，通过使用 COM 可连接对象、COM 消息服务的消息、CORBA 的事件或者 JavaBean 的事件），也可以通过构件框架做中介来间接通信，这时构件框架可以规范构件间的交互。当构件框架实例交互的时候同样的选择又会发生——这次的中介者是第三水平层次的运行实例。在图 9-2 中，CI 代表构件实例，CFI（Component framework instance）代表构件框架实例，CFFI（Component Framework of Framework Instance）代表构件系统（或者构件框架的框架）实例。

在单体软件处于主导地位的世界里，甚至第一水平分层的体系结构都是不常见的。值得一提的是，对象和类框架并没有形成最底层的水平分层。水平分层的结构是从可部

署的实体——构件开始的。传统的类框架只能形成单独的构件，独立于水平分层体系结构的布局。对象和类框架可以存在于构件内部。这些对象和类框架可以形成自己的层次，这取决于构件的复杂性，比如在 OLE 中的 MFC。但是，当编译构件的时候所有类框架的结构都会被展平。跟构件框架不一样的是，类框架和它的实例间的界限是很模糊的，这是因为这个框架在运行时刻并不是实体，而在编译时刻，实例并不存在。这种二重性也可解释我们对术语“类”和“对象”常见的混淆。

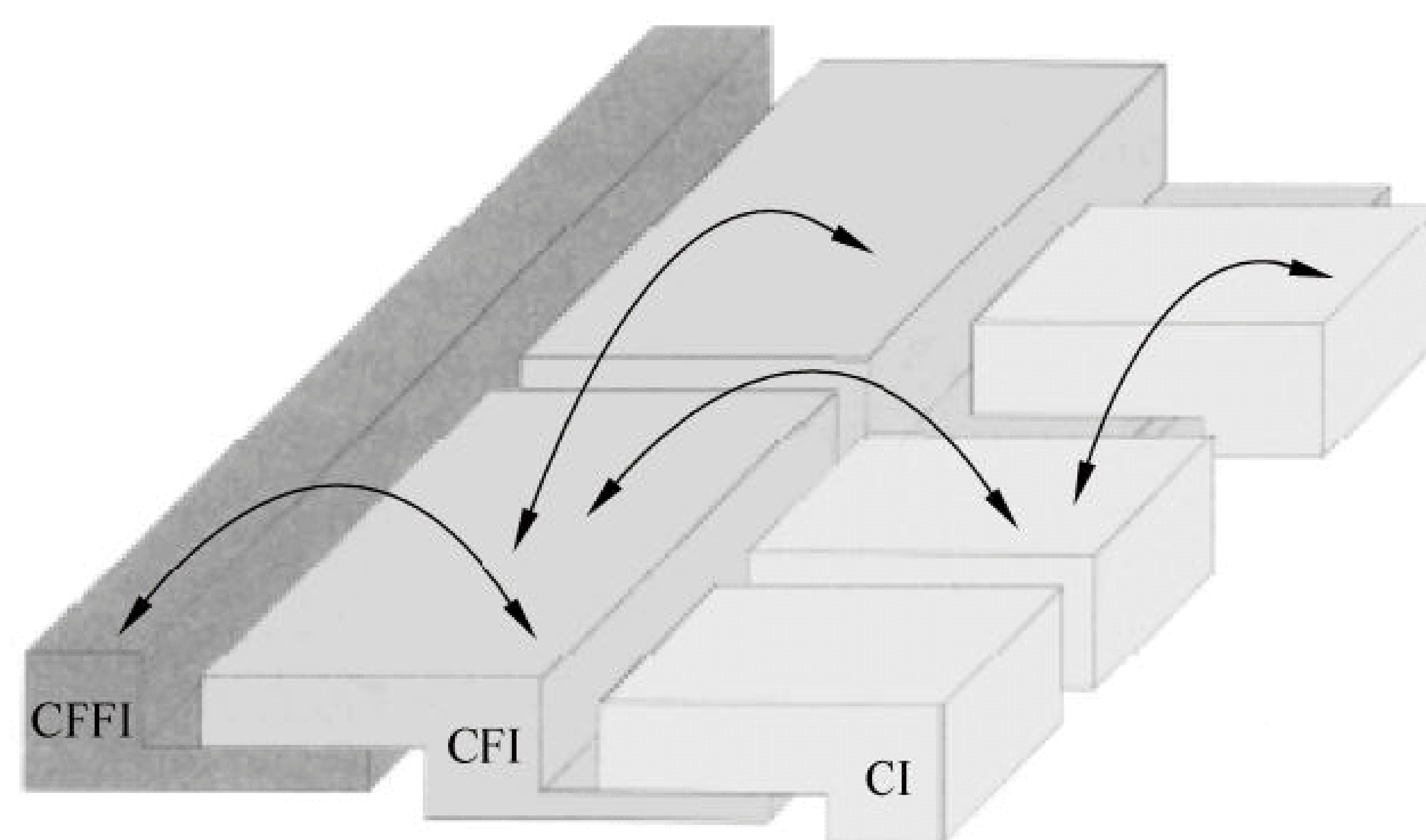


图 9-2　多水平分层体系结构中的自由与间接的交互

轻量级体系结构把注意力集中到一个问题，而不覆盖所有的问题，以有效支持轻量级构件的创建。这种构件的创建在一些限制性假设的前提下很容易想象出来。如果轻量级构件的指导性体系结构在限定其他决策的同时支持较好的易扩展性，那么它的商业价值就非常大。

5．构件和中间件

中间件是一个软件集合的名字，这些软件位于操作系统和高层次分布式编程平台之间。中间件有时被分为面向消息的中间件（Microsoft Operations Manager，MOM）和面向对象的中间件（Object Oriented Method，OOM）。然而，现有的大多数中间件都是这两种类型的混合体。当然，现在也有一种趋势是由传统的操作系统直接支持。操作系统总是包含了对通信协议的支持。Web 服务的推进和程序世界从以程序为中心到以协议为中心的转变，导致两种中间件的价值观：支持合适的协议或者提供简化本地服务构造的结构。

独立的中间件产品，如消息队列系统、事务处理监控器或者集线器，已经慢慢地消失了。取而代之的是结合了中间件功能和某个特定构件框架的特殊的服务器。应用服务器结合了应用管理、数据事务、负载平衡和其他的功能。集成服务器结合了协议转换、数据变换、路由和其他功能。工作流和复杂交互服务器结合了事件路由、决策和其他功能。

应用服务器市场有很多种不同的产品，包括 IBM 的 MQ 系列工作流系统和 Microsoft

的 BizTalk 服务器。集成服务器市场可能是最分散的，有各种提供商提供的各种类型的产品，包括 CrossWorlds，IBM（WebSphere B2B Integrator），Microsoft（BizTalk server），Oracle（XML lntegration server），Tibco（ActiveEnterprise），WebMethods（Enterprise）和 WRQ（Verastream）等。

6. 构件与生成式编程

生成式编程致力于通过转换的方法来构造软件。这种转换对软件工程师来说并不陌生。编译器把源代码转换成目标代码，JIT 编译器将中间代码变换成机器代码。然而，生成式编程试图超越传统的转换方法中转换器固定不变的弊端。其思路是允许程序员定义新的转换器。Czarnecki 和 Eisenecker（2000）详尽探讨了一个方法，它使用 C++模版来定义变换。他们同样也讨论了很多其他的方法，包括诸如 GenVoca 家族（Batory 和 O\Malley，1992）中的特殊的生成技术。而 Biggerstaff 则更广泛地讨论了生成式方法的动机（1998）。在可部署构件的世界中，生成式方法在两个领域里面起着重要作用。它们可以用来生产单独的构件，也可以用来增强由构件组装的系统。如果用于生产单独的构件，生成式方法就限定在单个构件中。当目标是生产规模较大的构件或者是生产潜在的数目较大的相关构件时，这个方法显得特别有用。仔细挑选可以被边界条件参数化的技术是很重要的，这些边界条件是对生成构件的需求。特别地，必须要精确控制实际的构件边界，包括提供接口和需求接口。此外，必须能精确控制同其他构件间的静态依赖。

9.3.2　语境相关组合构件框架

构件框架使构件化软件发展成为最重要的一步。当前大多数研究的重点都放在单个构件的创建和基本构件间的绑定支持。在这种条件下，独立开发的构件几乎不可能进行有效的协作。因而，构件的独立部署和集成将无法实现。

构件框架是一个软件实体，该实体支持符合某种标准的构件，允许这些构件的实例“插入”构件框架。构件框架为构件实例创建了环境条件、规定了对象实例间的交互。构件框架能单独存在并为某些构件创建其生存空间，构件框架也能与其他构件和构件框架协作。因而我们可以很自然地把构件框架自身建模为构件。这样，我们可以在构件框架的基础上建立更高层次的构件框架，该构件框架规定了其底层构件框架间的交互。

构件框架关键性的贡献在于体系结构准则的部分强制要求。在构件框架的控制下，通过强制构件实例执行某种任务，构件框架能够强制某些策略执行。我们来看一个具体的例子，一个构件框架可以强制规定事件多播时的某种顺序，这样就排除了一类由于误操作或竞争而导致的难以捉摸的错误。

第一个对语境相关组合提供商业支持的也许是 COM 的“套间”模型。事实上，其下一代，MTS（Microsoft Translation Server）语境，能看做是当前所有语境相关组合方法（EJB 容器、COM+语境、CCM 容器和 CLR 语境）的源头。顺便提一句，必须注意 EJB（Enterprise JavaBean）和 CCM（Component Category Model）容器紧密地对应于 MTS，

COM+和 CLR 语境，而不是对应于 OLE 和 ActiveX 容器。一个 OLE 和 ActiveX 容器并不截取所有内含的控件的输入或输出调用。

1．COM+语境

COM+源于 COM“套间”和 MTS 语境。COM“套间”使用线程模型来分离对象；MTS 语境通过事务域分离对象。COM+统一了这两个概念，同时也加入了大量的新的语境属性。无论哪种情况，用于驱动语境运行时语境的构造和将对象放置在合适的语境中的都是公开声明的属性。在 COM“套间”的情况下，属性声明采用每一个 COM 类都有一个注册表项的形式。注册的条目要求类的实例必须仅仅被放置在单线程“套间”中，或者仅仅被放置在多线程的“套间”中。（COM+增加了可租赁线程“套间”的概念，在这种类型的“套间”中一次只允许一个线程入住，但是多个线程能顺序地入住该“套间”。）

微软事务服务器（MTS）引入了事务语境的概念。在 MTS 中运行的 COM 类的事务声明属性规定，该类必须或者位于非事务语境，或者位于新的事务语境，或者位于新的或是已有的事务语境，或者不做任何要求。通过这些声明，MTS 和分布式事务协调器（Distributed Transaction Coordinator，DTC）共同创建了一个合适的事务域。相同事务域中的对象共享一个单独的逻辑线程和一个单独的共享事务资源集合（比如数据库连接和锁）。一旦线程从事务域中返回，事务要么提交要么终止，该事务域被销毁，该域所持有的资源被释放。

对于 MTS 的事务域，COM+增加了临时从一个事务域中返回、保持当前正在进行的事务的状态和在下一次调用时继续执行事务的功能。该模型允许客户端通过多个调用/返回接口进入一个事务服务的对话。（COM+也极大地扩展了语境属性、域和声明配置等概念。通过与以前的微软消息队列（Microsoft Message Queue，MSMQ））。服务集成以及加入几个其他的服务，COM+具有了大量的声明属性。COM+还引入了队列构件，这种类型的构件通过接收消息而实例化。

在 COM+中，如果两个构件共享一组兼容的语境属性集，则它们可以被看做是处在同一域中。比如，如果两个对象共享相同的事务 ID，则它们处于相同的事务域中。MTS 中的域能够扩展进程和机器的边界。而 MTS 中的语境自身对进程进行了划分，而不是扩展了进程边界。相同语境中的对象具有相同的语境属性。域是同一属性的分组，而语境则基于属性集。图 9-3 显示了三个语境，两个属于相同的事务域（c2 和 c3），两个属于相同的负载平衡域（c1 和 c2）。这两个域都含有相同的语境 c2。图中还显示了三个语境中的 4 个对象。对象 u，

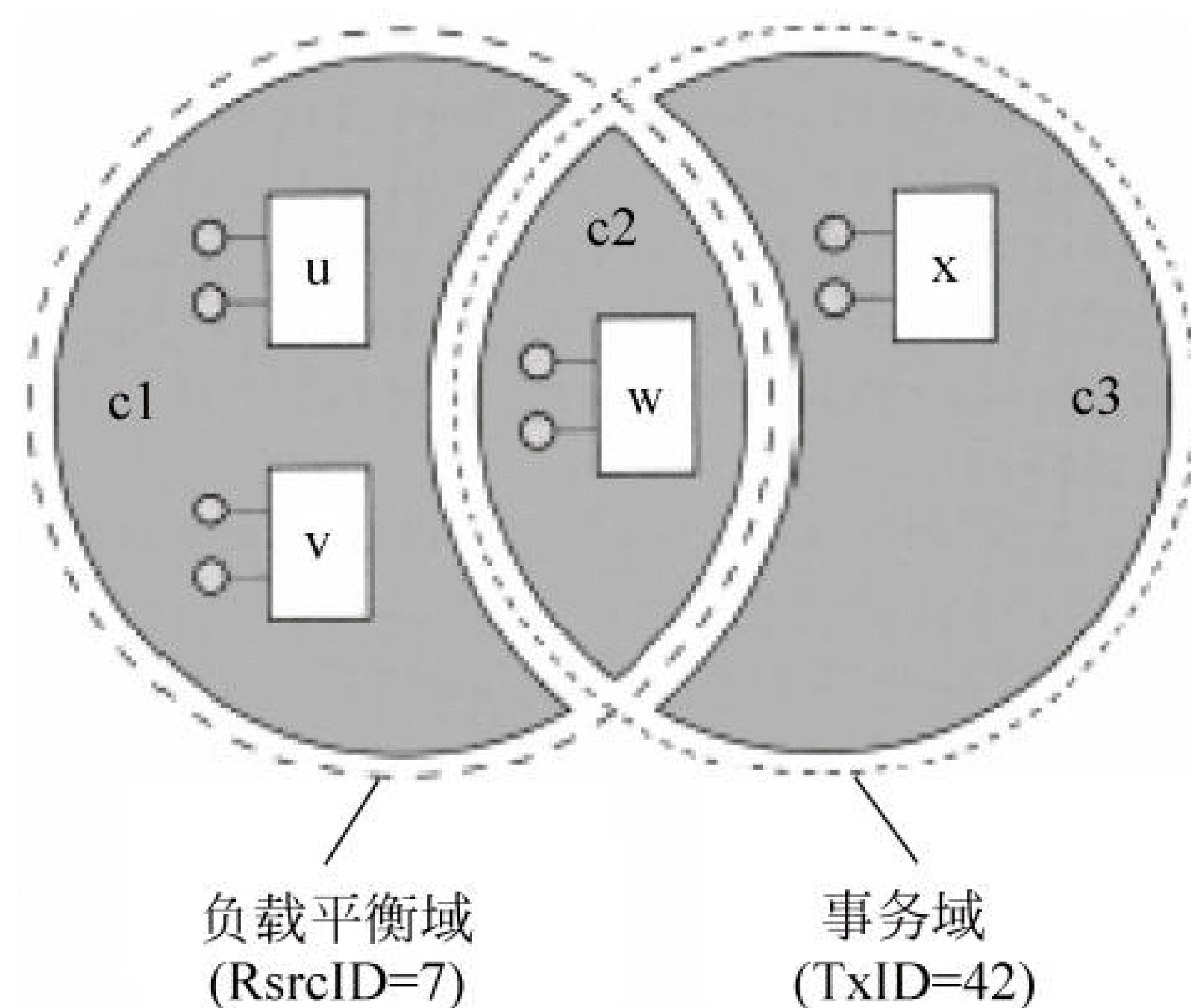


图 9-3 域和语境

v 和 w 共享相同的负载平衡资源（资源 ID 为 7），对象 w 和 x 共享相同的事务（事务 ID 为 42）。

跨越语境边界的调用被拦截，然后根据语境属性或者被预处理，或者被置后处理，或者被拒绝。

2．EJB 容器

EJB 为 EJB 实例提供了容器。虽然这些容器在 MTS 语境的基础上被模式化，但它们仍然有自己的变化。EJB 容器允许类请求进行明确的事务控制——也就是说，该类的实例需要直接调用事务 API 来开始、提交和结束一个事务。如此明确的控制使得这些类组装更加困难，但是当添加合法的事务代码时这种控制开辟了一条更加容易的道路。明确的控制在 COM+中是可能做到的，但是不如 EJB 直接（本质上，外部控制需要独立存在于 COM+的事务域之外，并且和微软的分布式事务控制器 DTC 直接发生交互）。

此外，EJB 容器通过支持会话 Bean 和实体 Bean 支持持久对象。会话 Bean 的行为类似于 MTS 语境中的 COM 实例——一旦事务结束（异常中断或是正常提交）其状态将丢失。实体 Bean 在事务正常提交时就保存至事务永久存储器。也就是说，Bean 能够作为永久对象保存，除了可显式编写语句——这是 MTS 或 COM+的唯一选择。

从版本 2.0 开始，EJB 提供了改进的由容器管理的持久性和关系，提高了 EJB 实体 Bean 从一个应用服务器的容器到另外一个应用服务器容器的可移植性。EJB 2.0 也增加了消息驱动的 Bean 类型，这是一个完全由数据驱动的 EJB 构件类型。与无状态会话 Bean 一样，消息驱动的 Bean 也是在消息到达时实例化，消息处理完毕后被销毁。这类似于 COM+中的队列构件。消息驱动激活的概念可以追溯到 IBM 信息管理系统（Information Management System，IMS）的事务管理功能，该系统激活和停止 IMS 程序以处理队列消息。

3．CCM 容器

CORBA 构件模型（CCM），CORBA 3.0 规范的语境中被引入。正如 EJB 建立在 MTS 的概念上一样，CCM 建立在 EJB 的概念上。CCM 容器可以定义为 EJB 容器的超集（EJB 规范和 CCM 规范分别发展的事实意味着，如果 EJB 不吸收 CCM 的进步、将二者融合的话，CCM 将赶上 EJB）。CCM 在 EJB 的会话构件和实体构件之外增加了对过程构件的支持。更确切地说是，CCM 的会话构件相对于 EJB 中的有状态会话 Bean，然而无状态的会话 bean 在 CCM 中被称为服务构件。过程构件的实例的状态在一次调用后不再保持。过程构件实例具有持久的状态但是不能通过主键来定位。因此，过程构件对于捕获正在进行的过程状态是有用的，但不能用于捕获可确认的实体状态。

4．CLR 语境和通道

CLR 语境的内部设施也许是第一个尝试为语境相关组装提供真正的可扩展设施的主流结构。它不像 MTS，COM+，EJB 和 CCM 容器，其所谓的语境属性的列表不是封闭的。如果语境中的对象确实需要，那么第三方就能够在语境边界上添加新的属性。这

样，当调用跨越语境边界时，语境的属性能够截取、产生作用或操作任何输入和输出消息。

当构造一个新的语境时，CLR 提供一个一次性的机会在语境边界上设置属性。一个特殊的语境属性可以通过编程的方式添加，比如通过其他的属性或是请求创建该属性的对象。另外，一个新的语境属性能够被创建它的对象公开地请求。公布的机制依赖于 CLR 的定制属性——能够直接被 CLR 类替换的可扩展元数据。类似于 COM+属性或是 EJB 的部署描述，这样的定制属性需要明确的声明。比如，一个类需要同步支持。当实例化时，系统检查这个新的对象是否处于一个具有同步属性的语境中。.NET 框架为 COM+企业服务定义了标准的属性。这样，通过 CLR-COM 之间的互操作内部结构，在 CLR 中调用 COM+的服务，创建被管理的对象就变得很容易了。

CLR 对象有 4 种类型——值类型、传值类型、传引用类型和语境约束类型。在通过应用域（AppDomain）的边界通信时，值类型和传值类型都是使用值编排。传引用和语境约束类型则都是将应用编排。其中唯一有趣的方面在于它们创建了一个方便的编排边界。编排在通道上执行，并能加入新的通道类型。标准类型包括在 SOAP/HTTP 协议之上编排和在 DCOM（Distributed Component Object Mode）之上编排。在通道的末端，如果系统为传引用类型的对象设置了代理则系统会重新构造传值对象。新的代理实现也可以被加入。

语境约束类型总是位于一个带有合适属性的语境中。位于语境之外的其他对象是语境“(agile)”。COM 中类似的概念叫“套间（agile）”对象，但这个概念不安全。不过，在 CLR 中，对语境约束对象的应用不能撤销。当调用通过语境边界时，语境边界拦截所有调用者或者被调用者是语境约束对象的所有调用。例如，当在一个“(agile)”对象的域中存储一个语境约束对象的引用时，先传递该引用到另外一个语境，当通过该引用调用对象的一个方法时，该调用将会被拦截。如果该引用——通过各种方式——回到了原来的语境中，则当另外一个调用到达时，该调用将不会被拦截。

默认的情况下，新的对象放置在和请求创建新对象的对象一样的语境中。或者，新创建的对象可以选择其他的语境（不过需要该语境中另一个对象的“帮助”）或者创建一个新的语境。如果被选中的语境的属性与该对象声明的要求不匹配，则一个新的语境将自动被创建。

5．元组和对象空间

在所有以上方法之前的一个关于语境相关组装的方法是基于无所不在的数据空间的概念，数据空间能够在不需要明确寻址的情况下被用于通信。该工作的发起人是耶鲁大学的 David Gelernter 和他的小组（www.cs.yale.edu/Linda/linda.html），他们建立了该设计领域。特别要指出的是，由 Nicholas Carriero 和 David Gelernter 创建的 Linda 协作语言引入了元组空间的概念。那些保存有原子数据的空间就称为元组。在元组空间上，Linda 仅仅定义了三种基本的操作——添加一个元组到一个空间中、在一个空间中匹配和读取

一个元组、在一个空间中匹配和删除一个元组。当前这种思想的“追随者”是 JavaSpaces（java.sun.com/products/javaspaces）。

协调数据和对象空间有一些附加的好处，比如我们没有必要确认构件实例之间的位置和它们的依赖性，决策安排也与数据流上的功能需求完全分开。同时，这些性质也面临着挑战。比如，为了避免在分布式实现中的集中瓶颈问题，元组空间不应该位于一个单独的物理位置。然而，为了实现高效的安排，有效的信息和被请求的数据元组或对象需要被传播至系统的每一个部分。保持元组空间操作原子性的需求使得对机构的要求进一步复杂化。

元组和对象空间对语境相关组装的求精有潜在的好处——即数据驱动的组装。在写这本书的时候，该方法已经不局限于研究项目，并影响着主流的技术。然而，有的人会说目录结构的广泛使用是元组空间融合的有趣的例子。这方面的例子包括因特网域名服务（DNS），轻量级目录访问协议（LDAP）目录，微软的活动目录和 Web 服务的 UDDI 目录。

目录假定了一个弱一致性模型，即条目很少被复制，临时的更新不一致性也是可以忍受的，条目改变的频率远远低于读条目的频率。这些性质保证了目录实现能进行大规模甚至是全局的可扩展性。比如，DNS 就被实现为 DNS 服务器的全局层次，这些服务器联合起来以惊人的速度来处理数百万台机器的基于 DNS 的名字解析请求。

可以证明，所有主流的构件方法都是依靠目录来进行某种形式的语境组装，而没有哪种主流方法是像 Linda 协作方法所建议的那样通过构件框架来做。相反，目录服务通过一些 API 得以应用，而协作的工作则留给客户端构件的开发人员。

更通用的数据驱动的构件框架也是存在的。在 COM+的队列构件或者 J2EE 的消息驱动 Bean 中，数据的分布就是通过面向消息的中间件（消息排队系统）来进行的。消息的到达引起相应的处理构件的自动激活，后者则通过在局部产生影响或者进一步发送消息来做出响应。

9.3.3　构件开发

面向构件的编程目前仍然是一门年轻的学科，其涉及的许多方面仍需要进一步地研究。本章的论述主要涉及了面向构件编程的方法学、环境和语言等三个方面。编程方法学主要考虑如何用一种系统化的方式来进行构件系统的划分、构件的交互和建造。而编程的环境和语言则主要考虑如何表现和支持特定的编程方法学。

1．面向构件的编程方法学

如同面向对象的编程（OOP）关注于如何支持建立面向对象的软件解决方案一样，面向构件的编程（Component-Oriented Programming，COP）关注于如何支持建立面向构件的解决方案。一个基于一般 OOP 风格的 COP 定义如下（Szyperski，1995）：“面向构件的编程需要下列基本的支持：

——多态性（可替代性）；
——模块封装性（高层次信息的隐藏）；
——后期的绑定和装载（部署独立性）；
——安全性（类型和模块安全性）。”

面向构件的编程仍然缺乏完善的方法学支持。现有的方法学往往只关注于单个构件本身，并没有充分考虑由于构件的复杂交互而带来的诸多困难。其中的一些问题可以在编程语言和编程方法的层次上进行解决。这其中，面向连接的编程尤其吸引了语言设计领域众多研究者的关注，例如，ArchJava（Aldrich 等人，2002 和 Jiazzi McDermid 等人，2001）。然而，面向连接的编程并不是通向面向构件的唯一途径。

分层体系结构或其他体系结构的设计方法有助于控制系统的复杂性，并能够指导系统的演化。但是，单独依靠体系结构并不能有效地指导构件及构件框架的开发活动。许多问题仍然没有得到根本的解决。主要问题如下。

1）异步问题

当前的构件互连标准大都使用某种形式的事件传播机制作为实现构件实例装配的手段。其思想是相对简单的：构件实例在被期望监听的状态发生变化时发布出特定的事件对象；事件分发机制负责接收这些事件对象，并把它们发送给对其感兴趣的其他构件实例；构件实例则需要对它们感兴趣的事件进行注册，因为它们可能需根据事件对象所标志的变化改变其自身的状态。

2）多线程

“多线程会使你寝食难安。”Swaine 在后来的著作中解释，他的一些与此相似的论断具有明显的煽动性，但是他并不认为这些论断是错误的。多线程是指在同一个状态空间内支持并发地进行多个顺序活动的概念。相对于顺序编程，多线程的引入为编程带来了相当大的复杂性。特别是，需要避免对多个线程共享的变量进行并发的读写操作可能造成的冲突。这种冲突也被称做数据竞争，因为两个或多个线程去竞争对共享变量的操作。线程的同步使用某种形式的加锁机制来解决此类问题，但这又带来了一个新的问题：过于保守的加锁或者错误的加锁顺序都可能导致死锁。

多线程主要关注于对程序执行进行更好的分配，发送并发请求的客户端能够很好地观察到这种分配。然而，获取性能最大化的手段却根本不依赖于多线程，而是尽量在第一时间内以最快的速度处理用户的请求。即使能够避免死锁，同步也可能导致一定程度的性能损失。必须避免对经常使用的共享资源进行不必要的加锁。跨线程的异常传播也会导致处理非同步的异常变得更加困难。而且，使用多线程和复杂的互锁机制将使得代码调试变得异常困难。

显然，在真正并发的环境下，这些问题无一不需要考虑。例如，如果构件实例运行在独立的处理器上，就需要考虑并发请求的问题。可以在处理一个请求时对某个构件实例进行完全的加锁，但这样做可能会导致死锁或者糟糕的响应时间。

3）“生活”在没有实现继承的状态下

构件间的实现继承所引起的严重问题使得人们倾向于使用简单对象组合或消息转发来替代实现继承。但是，当我们仅需要对已有的实现进行轻微的修改时，这种替代方式却又显得太笨拙。创建一个具有很多方法的类的子类，且只重写父类中一小部分方法是很容易实现的。相对而言，仅为转发一小部分方法调用而生成一个新的包装类却是一件十分繁琐的事情。除了实现上的开销以外，简单的转发还增加了运行时的开销（执行时间和代码占用空间的增加）。

当一个对象中的方法被分组成若干个接口，每个接口中含有数目恰当的方法时，COM 风格的聚合有助于避免由于转发所带来的性能损失。通过使用多种自动化技术，对于程序员而言，其实现成本没有丝毫的增加。

一种解决方案是根据转发目标对象的接口生成转发类的代码。这种方案的弱点也是所有代码生成途径所共有的：目标对象接口的改变要求重新生成转发类的代码，或者手工调整旧的生成代码。

另外一种解决方案是利用模板机制（如 C++中的模板）在编译时刻生成所需的代码。模板可以通过参数化的方式配置，而不必手工编辑生成的代码。编译器根据模板的实例化参数来生成最终的代码。

4）坚壳类

第三种解决方案是使用实现继承。虽然一般来讲实现继承具有严重的问题，但对于白盒类（以完整的源代码形式发布并且不再被改变的类）使用实现继承是没有问题的。对经常成为转发目标的构件接口来说，可以为其关联一个专门负责处理转发者琐碎细节的坚壳类（Szyperski，1992b）。坚壳类与转发目标具有相同的接口，而且所有方法的实现都是简单地向目标转发消息。坚壳类本身是抽象的，尽管所有的方法都有实现，但是这种实现完全没有引入新的功能。（有趣的是，某些语言包括 C++，无法表现这样一个事实，即一个没有抽象方法的类仍然是抽象类。）然而，可以通过继承坚壳类去截取某些方法调用，从而产生一个有意义的转发者。由此导致的程序员工作量与一般的实现继承类似，但产生的效果是转发而不是代理。

Java 中的代理类（proxy class）就是这样的一种机制。一个代理看起来是某种给定的类型，但其内部却实现为代理类的一个子类。这种实现提供了一种对调用进行截取的机会。CLR 通过实时代理类也提供了类似的机制。

5）语言支持

第四种解决方案是语言支持，这种解决方案或许更易于被接受。如果编程语言直接支持转发类的构造，则所有以上方案的缺点都可以被避免。编程的开销也将是最小的，且在运行时刻时间和空间上的开销与实现继承方式相比也都是一样的。但目前还没有主流的编程语言来支持这种构造。比如，C++的虚拟基类机制不允许在几个独立的对象之间共享基类对象。它也不允许动态改变基类对象，或虚拟基类的独立子类化。Objective-C

（Apple Computer，2000；Pinson 和 Wiener，1991）是一种支持对象动态继承的非主流编程语言。

6）调用者封装

语言支持带来的另外一个好处是接口定义。当构件对外提供一个接口时，可能会涉及两种不同的意图。一方面，构件外部的代码可能会调用这个接口中的操作。另一方面，构件内部的代码可能需要调用实现这个接口的一些操作。在 COM 技术中，这体现为入接口和出接口的差异。除了 Component Pascal 以外，没有别的语言能够恰当地支持构件的纯出接口。

正如许多传统的封装机制一样，如果出接口和入接口之间的对称可以被接受，那么仅对调用者而不是对被调用者进行封装也就不应该令人感到惊奇。然而，适合于构造构件的调用者封装机制被大多数的语言丢弃了。在类似 Simula 的语言，包括 Beta 语言（Lehrmann Madsen 等人，1993），均支持内部方法。在类的层次上，这种机制和调用者封装很相似。引介基类（Introducing Base Class）之外的代码无法访问由子类所实现的内部方法。每个人都可以试图调用该方法，但是基类代码的执行受到了保护，至少能够动态地防止非法的外部调用者。

调用者封装策略被应用于黑盒构件框架的若干方面。例如，只有框架可以调用视图的关键方法。如果框架在早些时候捕获了对同一个视图的同一个方法调用产生的异常，它将会阻止对此方法的进一步调用。这样，产生错误的视图的某些方法将被屏蔽，从而不会再继续扰乱系统的运行。黑盒是很少几个能够保证嵌在复合文档中的视图不会破坏文档的整体性的系统之一。

2．环境与选择目标框架

脱离了良好定义的环境，一个构件实例是不能正常工作的。构件框架定义了这样的环境。然而，一个构件实例可能被设计成可以在多个构件框架中工作。根据构件系统体系结构的不同，框架可以根据不同的角色被分割成不同的子框架。例如，每个框架都可能会采用某种特定的机制实现构件之间的协同运作。在这种情况下，分布式框架可能会负责在机器之间分发构件实例。而另外一个单独的框架将会负责对复合文档的集成。构件的设计可能需要考虑所有的这些子框架，以使得最终的构件实现能够在这些框架下正常运行。

3．工具与选择编程语言

原则上，构件编程几乎可以使用任何一种语言，并采用任何范型。并不存在所谓的最低需求。构件编程主要关注的是对相关构件的多态处理。由于构件之间的交互需要动态进行，因此就必须支持后期绑定。参数构造安全性还需要封装及安全——类型安全和模块安全——的支持，在大多数情况下垃圾回收的支持也是必需的。此外，构件编程需要一种能够显式化声明状态依赖的机制，理想情况下应该保持这些依赖是可以参数化的。对实现中的依赖进行完全的参数化导致了面向连接的编程。在语言范型的层次上，面向

对象范型最接近于面向构件的编程范型，但是其他的范型，比如功能范型，可能也是合适的。

到目前为止，只有少数编程语言在应用层次上支持面向构件的编程。许多流语言，如 COBOL、Object COBOL、FORTRAN、C、C++ Pascal 和 Smalltalk 在不同程度上均缺少对封装、多态、类型安全性、模块安全性的支持。

Java、C#和 Component Pascal 都分别支持在包一级或者模块一级的访问保护。通过这种方式，可以建立对模块安全性的支持。但 Java 的开放包机制对模块安全性的支持太过脆弱。即使不使用替换目标文件的方式，也可以通过向包中添加新的类，从而完全穿越包机制提供的保护！这种漏洞需要通过另外的途径来弥补。因此，需要把包放在文件系统中的保护目录下，或者其他带有访问控制的地方。在一个从远端服务器动态获取类文件的环境中，这种情况将变得更为复杂。需要一种机制来保证同一个包中的类文产生于同一个编译源。为了支持更加开放的设置，Java 或许需要采用封闭的模块构造机制，其中每一个这样的模块被映射到一个被发布的编译文件上。Java 的嵌套类机制有助于建立真正的所谓模块，但是 Java 缺少一个能够支持对一个类及其嵌套类的访问保护层。另外，由于 JVM 实际上并不真正支持嵌套类，Java 编译器不得不把嵌套类抽取出来，放入单独的类文件中。

C#令人感兴趣的地方在于其模块级访问保护适用于集合。C#的任何构造机制（事实上任何基于 CLR 的语言都适用）可以被打包成为一个集合。一旦被打包，该集合也就被加密了，从而避免对其的任意窜改，这也使得对集合内部的访问控制机制变得强大和有效。因此这种基于 CLR 的集合内部访问机制是目前为止最灵活的包概念。

9.3.4　构件组装

构件是可被第三方独立部署的基本单元。每个构件的部署过程之间不是相互孤立的，构件实例之间通常会在一个或多个构件框架的介入下发生交互。将构件组装成系统的一种显而易见的方法是通过传统的编程方式进行。然而，由于这种方式支持用较简单的方式生成大多数常用的构件系统（或由于其能够完全避免单独的组装过程），因此构件的适用范围和生存能力都大大地增加了。

1．构件初始化及互连

体系结构描述语言（ADL）即遵从了这样的思想：这些语言通常都把构件和连接子作为其核心的建模概念。组装因此表现为选择一组构件并通过适当的连接子将这组构件进行连接的过程。基于这种方式的组装过程实际描述的是被选择的构件的“实例”应该如何通过适当的连接子“实例”互连的过程。这个细节揭示的一个重要之处在于：构成一个组合体的基本元素是构件或连接子的实例而不是构件或连接子本身。例如，虽然概念上一个构件可以出现在两个组合体中，但实际上是这个构件的不同实例将出现在这两个组合体的若干实例中。

BML（bean markup language）是种由IBMalphaWorks实验室在1998年发布的针对JavaBean的构件组装语言。BML基于XML并针对JavaBean构件模型进行了定制。通过使用XSLT，可以从更抽象的系统描述中生成BML。实际上，BML自身与JavaBean构件模型已相当接近，能够支持Bean构件实例的创建、访问及配置等操作。为了支持配置，BML允许对Bean属性进行访问和设置。当提供了具有这种配置方式的Bean构件实例后，BML能够用来绑定Bean构件，使其作为监听者监听其他Bean产生的事件。BML既可以通过直接产生配置后的可运行子系统而被解释，也可以被编译而生成Java代码。BML解释器的基础是bean定制化框架。该框架也能够支持实现不同于BML的其他形式的bean配置和互连语言。

1）构件的可视化组装

构件实例的可视化组装方式能够有效地简单化组装过程。例如，JavaBean构件能够区分其实例的使用和构造阶段。一个bean因此可以表现出特定的外观（例如，一个类似建筑单元的图标）、行为（例如，一个可以和其他实例连接的句柄）和帮助信息（例如，针对特定人员的构件组装帮助文档）。在组装过程中，构件被实例化，实例通过统一的方式把其具有的出接口和入接口连接到相关的实体上。JavaBean和COM技术均支持这种一般方式的连接范型。

2）用复合文档取代可视化组装

在构件实例可见的情况下（通过提供一个可视化的用户界面），专业的构造器或组装环境可以和一般用途的软件开发环境相统一。而通过复合文档，构造和使用这两个不同的环境也可以自然直接地集成在一起（文档代表应用系统），对文档的编辑相当于构件（实例）的组装过程。在这样的系统中，构件的组装者和使用者之间不存在任何的隔阂。这两者之间的平滑过渡就如同在已有构件组合体的基础上，通过后期组装生成新组合体，与通过编程生成新组合体可以随意地相互结合使用，以满足特定应用系统的需要一样。因此，为了全面地满足使用者的需求，组装机制应该具有在使用时刻的可用性。黑盒构件构造器及构件框架即遵循了这样的途径。虽然我们可以通过不部署所需的构造器构件的方式来区分构件的组装和使用，但复合文档并不对这两者做严格的区分。

构造环境和使用环境的无缝集成（特别是在复合文档的方式下）也形成了对于快速应用开发（Rapid Application Development，RAD）的强有力支持。在这样的环境中，工业级构件、原型构件及一次性解决方案可以自由地结合。需求捕获和对需求改变请求的确认也可以高效且有效地执行。如果企业或组织需要，经过相应的培训后，最终用户可以进一步地调整他们的系统。

3）非图形用户界面环境的构件

大多数早期的构件化软件方法往往关注于客户端的前台交互式应用系统。现代图形用户界面的需求本质，加上用户界面的相对规则性，使得与用户界面相关的可重用构件成为具有独特价值的软件资产。然而，计算的其他领域，特别是基于服务端的解决方案，

存在同样甚至更多的复杂性，而且已经引起了现阶段许多构件化软件方法的关注。

对基于服务的构件，构造和使用阶段的清晰划分显得更为自然。Oliver Sims 于 1994 年提出的业务对象，是最早提出的针对“无处不在的构件”（Components Everywhere）思想的建议之一。接下来的一个重要进展则是 Java servlet 的出现。Java servlet 是运行于服务器上的构件，但它们仍可以通过可视化的方式来进行组装。为了与现存的构件模型良好协调（包括 CORBA 提出的构件模型），使用前的组装通常需要较早地做出某些决策，如分布式系统中构件实例的分布决策。值得注意的是，虽然对象迁移能够带来的益处仍然值得讨论，但大多数系统，包括当前的 CORBA 实现，都支持持有其他对象引用的对象的迁移能力。然而最近的一些围绕 Web 服务的途径（基于 SOAP），却不支持对远程对象引用的传递。相反，SOAP 主张对定位器（Locator）的传递，如 URL 或 COM 中的 moniker。这些定位器在不同的机器上每次可能会被解析到不同的对象上。因此，远程对象标志这个概念在 SOAP 和 Web 服务中并不存在，迁移问题也由此变得非常简单。

关于服务端构件模型的典型解决方案包括适用于应用服务器的 EJB 模型（Sun 公司 J2EE 的一部分）和 COM+模型（微软公司），以及适用于 Web 服务器的 servlet 模型（基于 Sun 公司 JSP 技术）和 Visual Basic 及其他技术（基于微软公司 ASP 技术）。微软的.NET 框架还引入了一种新的同时适用于客户端和服务端的基于 CLI（Command Line Interface）的构件模型。

4）可管理且“自引导的”构件组装

构件的组装实际上是指对构件实例的组装（一个采用对象技术实现的构件实例通常是一个由若干对象形成的消息网络）。当然，通过组合已有构件来实现新构件也是可行的，这种方式类似于传统的基于底层函数库构建高层函数库的过程。换而言之，构件组装（不是构件实例）只不过是编程的一个代名词，而构件实例的组装并非如此。构件实例组装提倡把实现构件的代码和资源与“连接”构件实例的代码这两个方面保持分离。构件实例的连接可以通过轻量级编程的方式（如编写脚本）来实现，而新构件的编写则应采用其他的方式（脚本语言或接口语言并不适合编写构件，因为编写构件与连接构件实例有本质的不同）。

5）最终用户组装

当需要时，允许最终用户进行系统组装以获得高度定制的解决方案是非常有价值的。最终用户的参与导致产生了一个有趣的、介于完全自引导和完全静态预定义之间的构件组装模式。显然，即使有最终用户的参与，组装过程仍然需要一定程度的自引导性。我们不应当期望用户能够完全承担技术细节层次上的构件组装工作。

系统组装分为三个不同的层次：定制（customization）、集成（integration）和扩展（extension）。这三个层次对应于构件组装过程中的不同任务。但这种最终用户剪裁仅仅是从用户的角度观察其关心的领域问题，而不是从构件组装的技术角度来进行的。Robert

Slagter 和 Henri ter Hofte（2002）展示了这种思想在计算机支持的协作应用软件系统中的一个有趣应用。该系统支持最终用户根据需要去组合群件的行为。Groove Transceiver（www.groove.net）具有的类似特征则允许最终用户通过选择和配置工具快速地组装工作空间。

6）构件演化

构件技术体现了一种后期组装的思想。构件的逐渐成熟会进一步推后组装（或绑定）时间，但随之而来的是整个系统将变得越来越脆弱。构件通常也会经历一般软件产品具有的演化过程。安装新版本的构件将会与期望使用旧版本构件的现有系统发生冲突，甚至直接与现存的旧版本构件实例发生冲突。相对于已经实例化的构件，一个构件从构件库中被获取并实例化的时间越晚，潜在的版本冲突问题就会越严重。

在分布式系统中，为安装新版本的构件实例而终止所有现有构件的运行是不现实的。不同版本的客户端和不同版本的构件实例之间的二进制互操作性需要在版本间二进制兼容性中就加以考虑。如何实现构件实例的在线版本升级仍然是一个非常活跃的研究领域。

在实际配置中，必须考虑构件的不同版本实例共存于一个系统的情况。系统的升级就是一个重要的例子。除采用多版本共存技术之外，解决“遗留系统移植”问题还需要通过使用包裹器构件来适配旧版软件或解决系统不兼容性。

支持版本共存和包裹器构件技术的方法之一是 COM 所使用的方法。按照约定，一旦 COM 接口被发布，其就不能再被更改。因此，COM 中不存在针对单个接口的版本问题。一个提供新版本服务的构件将不得不使用一个新的接口。采用这种方式的一个重要优点是它使得同时支持具有不同语义的新旧接口成为可能。显然，一旦一个接口不再被支持时，该接口就可以从系统中安全删除。

CORBA 采用的版本管理机制的能力则较弱，它仍然试图将所有版本的所有操作合并成一个接口。因此，这种方式不支持仅改变操作的语义而不改变该操作名称或原型的能力。由此导致的一个后果是，虽然可以引入操作的新名称，但若想删除一个旧的操作而不改变二进制兼容性却非常困难。甚至 SOM 的版本序列机制（SOMs Release Orders）也无法解决这个问题。

第 10 章　构件平台与典型架构

几乎没有构件能独立的部署，它们大多数依赖于特定的基础设施平台。由于行业高度竞争，公用构建基础设施目前只有 CORBA+Java 和 Microsoft COM+CLR 两大阵营。尽管只有两大阵营，SOA 技术也飞速发展，不同平台构件连接能力有了一定改善，但在设计、管理、规范等方面存在很大差异。因此，我们有必要了解这些平台特点和差异，为应用开发选择合适的构件开发平台。

10.1　OMG 方式

成立于 1989 年的对象管理组（OMG）是目前计算工业中最大的组织。作为一个非营利性组织，OMG 旨在通过规范化对象开放市场的所有层次上的互操作性。至 2002 年，有近 800 成员加入 OMG。

10.1.1　对象请求代理

CORBA 的主要目标就是使不同语言、不同实现和不同平台间能进行交互。因此，OMG 从没有停步在“二进制”标准上（可配置、可执行级的标准），而是保证每个细节都被标准化，使其能顾及不同的实现及 CORBA 兼容不同产品的独立供应商增值的需要。这一开放式方法的不利一面就是 CORBA 兼容产品不能在二进制级进行有效的互操作，只能以较高的代价在高层协议上协作。OMG 的跨 ORB（对象请求代理）协议——IIOP（Internet InterORB Protocol）互操作协议，在 1995 年 7 月的 CORBA 2.0 中被规范化。与 ORB 的互操作兼容则必须支持 IIOP。在 1996 年 7 月的 CORBA 2.0 更新版本中，加入了一条关于相互作用的协议，该协议明确了基于 CORBA 的系统与基于微软 COM 系统之间的互操作细节。

CORBA 包括三个基本部分：一套调用接口、对象请求代理（ORB）和一套对象适配器。面向对象操作的调用实现后期绑定。对象引用所指代的对象实现决定了被调用方法的最终实现。调用接口支持不同级别的后期绑定，同时编排调用参数，使 ORB 核心能定位接收对象，调用方法，以及传递参数。在接收端，一个对象适配器还原参数，调用接收对象相应的方法。图 10-1 简单地描述了基本的 CORBA 结构。

10.1.2　公共对象服务规范

现有的 CORBA Service 包含 16 种对象服务（CORBA 服务），其中的通告服务是电信领域设施正式的组成部分。这些服务划分为两大类：一类服务应用于企业计算系统。

这些系统往往将 CORBA 对象视为模块，并视 CORBA 为易用的通信中间件，此时的 CORBA 服务大多用来支持大规模的操作；另一类服务则应用于细粒度的对象操作，但目前这些服务的实用价值较差。CORBA 3.0 中的持久状态服务（Persistent State Service，PSS）可能是一个例外，它替代了 CORBA 2.0 中的持久对象服务（POS）。PSS 是 CORBA 构件模型的三个主要支撑服务之一，另两个是事务服务和通告服务。值得注意的是，大型基于 CORBA 系统往往只使用少量的 CORBA 服务，包括名字服务、安全服务、事务服务和交易服务。现有大部分 ORB 产品并不试图支持全部的 CORBA 服务也说明了这一点。

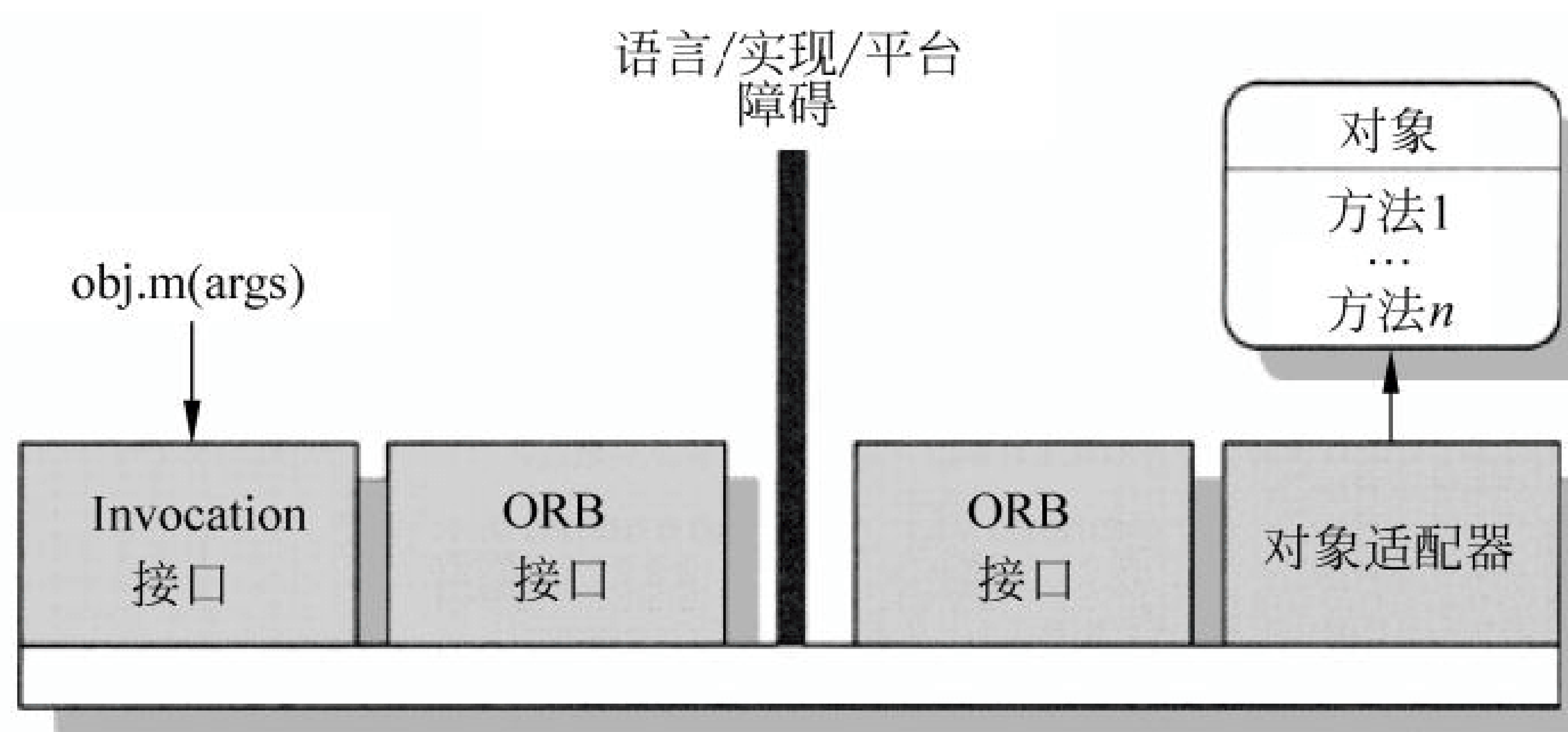


图 10-1 基于 ORB 系统的简化结构

1. 支持企业分布式计算的服务

许多大型企业系统只是将 CORBA 作为对象总线，依靠 ORB 与其他各种各样的系统进行互操作。名字服务是关键服务之一。

1）命名服务，交易器服务

每个对象内部都有唯一的标志符。命名服务则允许任意地给对象赋予一个名字，这个名字在其所属的命名语境中是唯一的。而命名语境所形成的层次结构，使得所有的名字形成名字树。

交易器服务允许给对象赋予一个复杂的描述，从而允许客户基于该描述来定位所需的对象。交易器通过交易语境来组织对象。客户则在指定的交易语境中根据对象描述的部分内容或关键字来搜寻对象，而搜寻结果往往是一个包含满足查询条件的一组对象的列表。OMG 交易器服务规范同时被 ISO（ISO/IEC 13235-1）和 ITU（ITU-T 推荐 X.950）所采用。

2）事件服务，通告服务

事件服务允许定义那些从事件生产者被发送到事件消费者的事件对象。由于信息只能从生产者流向消费者，因而事件对象是不变的。事件必须通过事件通道传播，从而松散了生产者和消费者之间的耦合关系。事件可以具有类型（使用 OMG IDL 定义），而通道可以根据类型过滤事件。

事件通道支持“推”和“拉”两种方式的事件通告模型。在“推”模型中，事件生

产者调用事件通道的“推”方法将事件上传给事件通道，事件通道进而调用所有注册的事件消费者的“推”方法将事件传给消费者。在“拉”模型中，事件消费者调用事件通道的“拉”方法，这将导致事件通道调用所有注册的事件生产者的“拉”方法，此时获得的新事件将返回给消费者。

1998 年，通告服务为事件服务增加了几个重要的特征——服务质量（Quality of Service，QoS）规范和管理；标准的类型化和结构化事件；基于类型和 QoS 的动态事件过滤；作用于资源、通道、一组消费者或单个消费者的事件过滤；针对资源、通道和客户的事件发现。值得注意的是，通告服务本身并不是 CORBA 服务，而是电信领域工作组（TelDTF）提交的 CORBA 设施。

3）对象事务服务

对象事务服务（Object Transaction Service，OTS）是建立分布式应用最重要的服务之一。OMG 于 1994 年 12 月制定的 OTS 规范在大多数的 ORB 产品及若干 J2EE 服务器中得到支持。OTS 实现必须支持平坦事务，而嵌套事务是可选的。遵循 X/Open 分布事务处理标准的其他事务服务可以与 OTS 集成。同样，多个异构 ORB 提供的事务也可集成。

在基于构件的系统中，嵌套事务似乎不可避免。因为一个构件的实现可能创建一个覆盖一系列操作的事务闭包，而这些事务属性无须在构件接口中声明。这种独立扩展性原则需要嵌套事务的支持。作为唯一的 OTS 实现必须支持的事务类型，平坦事务在构件系统中的价值有限。实际上，现有的主流事务中间件也不支持嵌套事务，这是它们共同的缺点。

OTS 自动维护当前的事务语境，该语境将随请求在 ORB 系统中传递，也可传递给其他的非 CORBA 的事务系统。对于 CORBA 对象，事务语境可以传递给任何实现了 Transactional Object 的对象。当前的事务语境可从 ORB 获得，因此必须保证随时可用。事务操作，如 begin，commit，rollback 都在当前的语境中定义。

所有希望在一次事务中执行修改，或者需要执行事务控制的对象都必须向 OTS 协调器注册。该协调器可从当前语境中获得。一个资源可以指明它是否支持嵌套事务。任何资源都必须实现 Resource 接口，从而允许协调器执行两段提交协议（众所周知，两段提交协议在完全分布的环境下可能发生死锁，这只能通过特定的广播协议避免；三段提交协议能够避免死锁问题（Mullender，1993），但开销太大。因此，OTS 规定协调器在逻辑上必须是集中式的）。

OTS 的设计目标之一是希望将事务控制作为一个独立的服务，但目前更普遍的是将事务和其他服务集成到应用服务器提供的语境或容器中。

4）安全服务

可靠的安全服务对于一个跨越多个相互信赖的组织的分布系统极为重要。安全服务必须得到普及。所有可互操作的 ORB 或可共同工作的系统必须协作，而这要求为所有

的参与者建立统一的安全策略。

CORBA 安全规范定义了一系列的服务，包括认证、安全通信、证书委托（也称为“替身”）及防抵赖等。目前仅有少数产品完全支持该规范，如 BEA 的 WebLogic 和 IBM 的 WebSphere。很多产品仅仅依赖 Netscape 的安全套接字（Secure Socket Layer，SSL）实施安全保障，尤其采用独立 ORB，而不是完全集成的应用服务器时。因为利用 SSL 可以很方便地实现简单的认证及安全的通信，但不能支持类似委托和防抵赖等较高级的安全机制。

5）支持细粒度对象互操作的服务

尽管有些服务，包括收集、外部化和查询服务，仍未被任何产品实现（原因很多，如查询服务的规范过于松散，收集服务的某些假设不切实际，本节仍将介绍余下的服务，以便读者对 OMA 涉及的对象服务有一个全面的认识。

6）并发控制服务

该服务支持对资源进行加锁和解锁。锁必须依赖于事务的语境或其他语境才能获得。依赖事务语境创建的锁将作为事务回滚的部分被释放。锁具有不同的模式，如读锁、写锁、升级锁。其中，读锁允许多个客户同时执行读操作；而写锁保证只有一个客户才能执行写操作；升级锁是可以升级为写锁的读锁，支持互斥的读操作。锁有多个锁集合。每个受保护的资源都拥有一个锁集合，该集合决定了可用的锁的种类及数量。一个锁集合的工厂接口支持创建新的锁集合。锁集合不是事务型就是非事务型的，并可与其他锁集合建立关联。锁协调器可以释放指定锁集合中所有的锁。

7）许可服务

组装构件的过程中需要获取所有非免费构件的使用许可。许可服务支持多种类型的许可模式。该服务定义了两个接口（抽象）——许可服务管理器和特定于厂商的许可服务。如果一个对象与一个许可协议绑定，那么它可以通过许可服务管理器检查其使用是否合法。

8）生命周期服务

这类服务支持创建、复制、移动和删除 CORBA 对象及其相关的对象组。下面将介绍如何利用关系服务提供的包含与引用关系来处理对象组。包含关系支持嵌套复制，即所有被包含的对象都会被复制。为了支持对象创建，生命周期服务支持注册与获取工厂对象。一旦获得所需的工厂对象，就能够用它来创建新的对象。

生命周期服务允许删除对象或对象组，但并不提示何时销毁该对象。这意味着分布式内存管理需要高层应用的参与，这被认为是 CORBA 较为明显的缺点。相比较而言，DCOM 支持分布引用的计数，Java 和 CLR 甚至支持基于租借的远程引用的分布式垃圾收集。

9）关系服务

关系服务指允许定义和维护对象之间的关系。不依赖语言级的指针或引用，该服务

引入了一种关系模型，以支持在不影响相关对象的情况下创建对象间的关系。但是，关系服务基本上没有实际应用，甚至没有产品实现，极有可能被基于 CCM 的业务对象关系所取代。

10）持久状态服务

持久性是指对象在其创建程序终止后仍然存活。为此，CORBA 2.0 制定了持久对象服务（POS），用来支持 CORBA 对象的持久性。尽管在 1994 年年初就被 OMG 标准化为关键服务，但直到 1996 年年中才出现一个 beta 版的实现。一些报告甚至指出该规范及其与其他对象服务的互操作存在严重的技术问题。另外，POS 没有解决“正确性”的问题，尤其是它把存储的申请交给应用代码处理。POS 规范最终在 CORBA 3.0 版本中被新的持久状态服务（PSS）所代替。

11）外部化服务

这项服务支持对象网和对象流之间的双向映射。对象网外部化后再内部化意味着创建该对象网的副本。外部化服务并不保证引用的完整性，仅保留同时外部化的对象之间的引用。外部化使得对象网的值复制成为可能。而外部化对象所需的其他对象的引用可保存为 ORB 为对象引用提供的字符串标志。

对象必须实现 Streamable 接口才能被外部化。为了外部化一个 Streamable 对象，必须首先调用实现了 Stream 接口的某个对象的外部化方法，该方法将调用流对象的 externalize_to_stream 方法，并传递一个实现 StreamIO 接口的对象。最后，流对象将任何 OMG IDL 定义的数据类型或实现写入 streamIO 对象。流对象也可以外部化由关系服务定义的整个对象图表。

12）属性服务

这种服务允许将任意的属性与对象关联起来，被关联对象必须实现 PropertySet 接口。属性可以独立地或成组地添加、获取和删除。如果一个对象还实现了 PropertySetDef 接口，则可按以下 4 种模式中的任一种进一步控制属性，这 4 种模式是：标准属性（可以修改和删除）、只读属性（能被删除但是不能修改）、强制标准属性（能被修改但不能删除）和强制只读属性。

属性服务并不说明属性的语义和内容。一般而言，对于程序有用的属性都需要由程序显式地赋予相应的信息。作为一个重要的例子，系统管理工具被赋予“粘贴”特性来有效地跟踪对象。

13）对象查询服务

该服务用来依靠属性定位对象。该服务类似于对象交易服务，但该服务定位对象实例而不是定位服务器。查询使用的属性由对象公布或者允许通过操作获得。有两种查询语言可供选择，面向对象数据库管理组的 ODMG-93 对象查询语言（Object Query Language，OQL）和扩展的 SQL。一个更为普遍的查询语言正在建立。

查询服务定义了其自身的一个简单的集合服务——是通用集合服务的子集。查询结

果集返回给用户时会用到集合。这些简单的集合提供了有序集的语义，包括增加或删除元素和元素集的操作。服务提供了一个 Iterator 接口来支持对集合元素的遍历。

14）对象集合服务

对象集合服务支持各种抽象拓扑集合，例如，包、集合、队列、表、树、角色模型是 Smalltalk 集合类库（Goldberg 与 Robson，1983，1989）。CORBA 的集合服务（基于 CORBA 对象的相对重权模型）是否可与本地的对象集合库竞争是一个有争议的问题，另外，对象库可能更适于在 ORB 间传输各种形态与属性的集合。

15）时间服务

这一服务处理拥有众多异步时钟的分布式系统固有的误差问题。许多应用程序中，用实时信息将内部事件（如创建文件）与外部通用时间建立关联。一个时间服务必须在允许的误差范围内实现这种关联，并避免其他非因果的关联。例如，假设一个新对象的产生是对另一个对象触发某事件的反应，那么，如果给前一个对象赋予“生成日期”的时间戳，而该时间戳却先于后一个对象产生的时刻，此时就会产生一个非因果的时间信息——这恰恰是非因果时间服务的典型结果。

10.1.3 CORBA 构件模型

CORBA 3.0 是 CORBA 标准中最新的一个。尽管 2002 年 6 月左右该规范最后部分仍未定稿，但针对 CORBA 2.0 全面的改进已经获得了显著的进展。除了对象服务的全面修订，最主要的成就恐怕就是新的 CORBA 构件模型（CCM）——尽管最终的 CCM 规范的发布仍未定案。（有时，CCM 也被称做 CORBA 构件。）

1．可移植对象适配器

CORBA 对象适配器主要的作用就是在一个 ORB 和真正接收调用并且返回结果的对象实现之间进行协调。目前采用的对象适配器的规范针对可移植的对象适配器，它代替了已过时的基本对象适配器。目前还没有其他的对象适配器规范。这种可移植对象适配器的一个实例为一组对象接收请求。任何 ORB 支持的服务器进程至少有一个 POA（Portable Object Adapter）的实例，当然，该进程中的每个服务对象都可能有一个 POA 实例。

一个 POA 实例通过将收到的请求传递给一个“服务体”来对其进行处理。“服务体”是 CORBA 对象的实现。图 10-2 给出了一个典型的工具使用场景，从一个 IDL 定义开始，客户端的指代，服务器端的 POA 骨架，服务器端的“服务体”模板被一一建立。开发者可以通过完成该模板来补充实现细节。CORBA 对象不强制使用面向对象语言，因而“服务体”也不一定为“类”。如果使用了面向对象语言，那么“服务体”就是类的实例。

2．CCM 构件

一个 CCM 应用程序是 CCM 构件的一个组装，其中构件可以是客户创建的或者是现成的、企业内部的或者是后来获得的。企业级 JavaBean 构件和 CCM 构件能够在一个应用程序中集成在一起。单个构件通过构件包发布，该构件包含有一个描述其内容的 XML

文档，还可以包含支持不同平台的二进制代码。CCM 的装配包含一个描述它们所引用的构件包信息的 XML 文档，以及它们的部署配置。

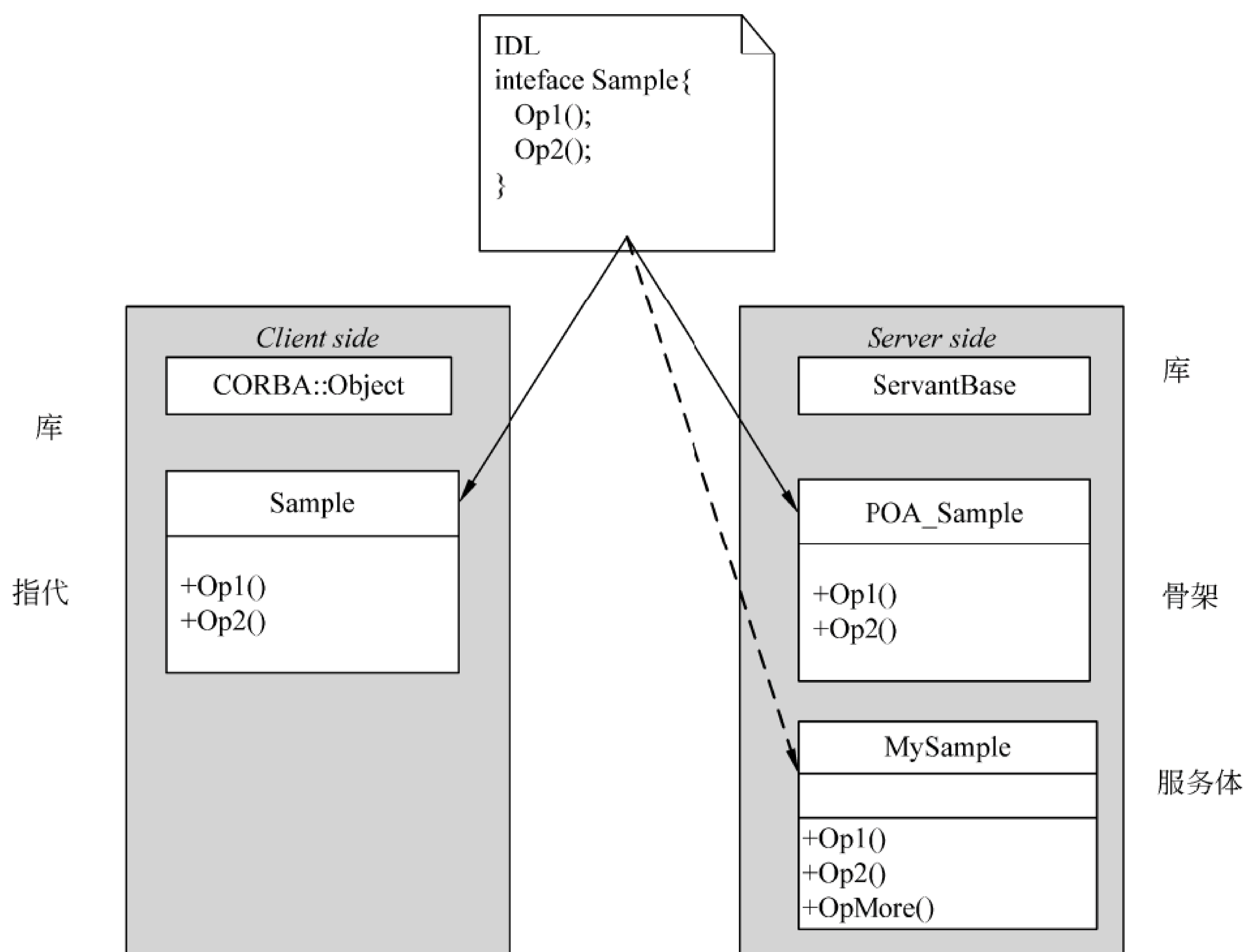


图 10-2　根据 IDL 文件生成指代、POA 骨架、服务体模板

3．CCM 容器

CORBA 3.0 定义了一个构件实现框架（Component Implementation Framework，CIF），其中包括接收 CIDL（Component Implementation Definition Language，构件实现描述语言）输入并产生实现代码的生成器。另外，每个构件实例都放在一个 CCM 容器里。构件通过容器的接口与 POA、事务、安全、持久化及通知服务相作用。一个容器同样也有插座接口来接收对构件实例的回调。

10.1.4　CORBA 设施

CORBA 设施可以分为水平的（普遍的）和垂直的（特定领域的）支持。不管是哪种支持，每个 CORBA 设施都定义了一个特定的构件框架，从而能够集成构件。最初，OMG 试图标准化 4 个领域的水平设施：用户界面、信息管理、系统管理和任务管理。但是这些努力都失败了，而且水平设施在今天的 OMA（Open Mobile Architecture）中的影响力很弱，之所以保留下来，是因为垂直设施的工作很可能产生并不特定于单个领域

的设施。水平设施的例子，或者已经被标准化或者正在考虑之中，包括全球服务、移动代理、时间和打印设施。

领域任务组定义了垂直设施。在2002年年初，有10个这样的任务组：商业企业集成命令控制、计算机通信和集成、财政、卫生保健、生命科学研究、制造业、空间、电信、运输和共用设施建设。

10.2 SUN公司的方式

10.2.1 Java构件技术的概述

就像上面说到的，Applet是Java中最初引起广泛关注并取得突破的地方。事实上，Java最初为了使不可靠的并可下载得到的Applet能够在客户端浏览器的进程中执行，在很多地方进行了特别设计，因而，不会造成无法接受的安全隐患。为实现这个目的，在Java中，编译器会检查Applet代码的安全性。这个做法的指导思想是：一个通过了编译器检查的Applet代码不会带来安全隐患。由于编译得到的字节码仍然可能被人修改，代码在装载时刻会被再次检查（称为“校验”）。通过校验的Applet是安全的，并受强制安全策略的约束。这一点对于现有的包括C++和对象Pascal在内的绝大多数编程语言来说都是不可能实现的。当然，安全策略可以在Smalltalk或者Visual Basic这些解释执行的语言中得到加强。然而Java是为允许编写在目标环境下有效执行的代码而设计的。这是通过所谓的“即时编译器（JIT）”实现的。

1．Java与Java 2

虽然最初Java的规范集深受Applet思想的影响，Java 2平台（在1998年后期发布）打破原有框架，并将Applet改变成一个边缘的角色。Java 2引入了平台版本的概念，从Java规范集中选出，并共同服务于一组特定用户关心的问题。图10-3给出了Java 2的组织形式。更多的关于Java 2的内容将在下面的章节中介绍。作为Java标志性的平台版本，J2EE（Java 2平台企业版）最初于1999年年底发布，并获得了巨大成功。J2EE是一组以EJB为核心的规范，在这些规范之下是由许多不同厂商提供的应用服务器（其中最大的两个厂商是提供WebSphere产品的IBM公司和提供WebLogic产品的BEA公司；到2001年年底，在Flashline.com评测比较表中列出了40个左右的厂商，它们的产品的价格从免费/开放源码到每个CPU 75 000美元不等）。微型版本J2ME也相当成功，特别是在用于移动电话的部分，企业版为构件化软件提供了丰富的环境。

除了版本之外，Java 2还区分了运行环境（Runtime Environment，RE）、软件开发工具包（Software Development Kit，SDK）和参考实现。运行环境是Java虚拟机和必须具有的J2SE API的实现。运行环境一般与一个SDK的版本相对应，SDK提供包括编译器和调试器在内的开发工具。容易混淆的是，按顺序为Java 1规范编号的1.x编号被继

续用来给 Java 2 的运行环境和 SDK 编号。所以，运行环境可以这样提出“Java 2 运行环境，标准版，v1.4”。

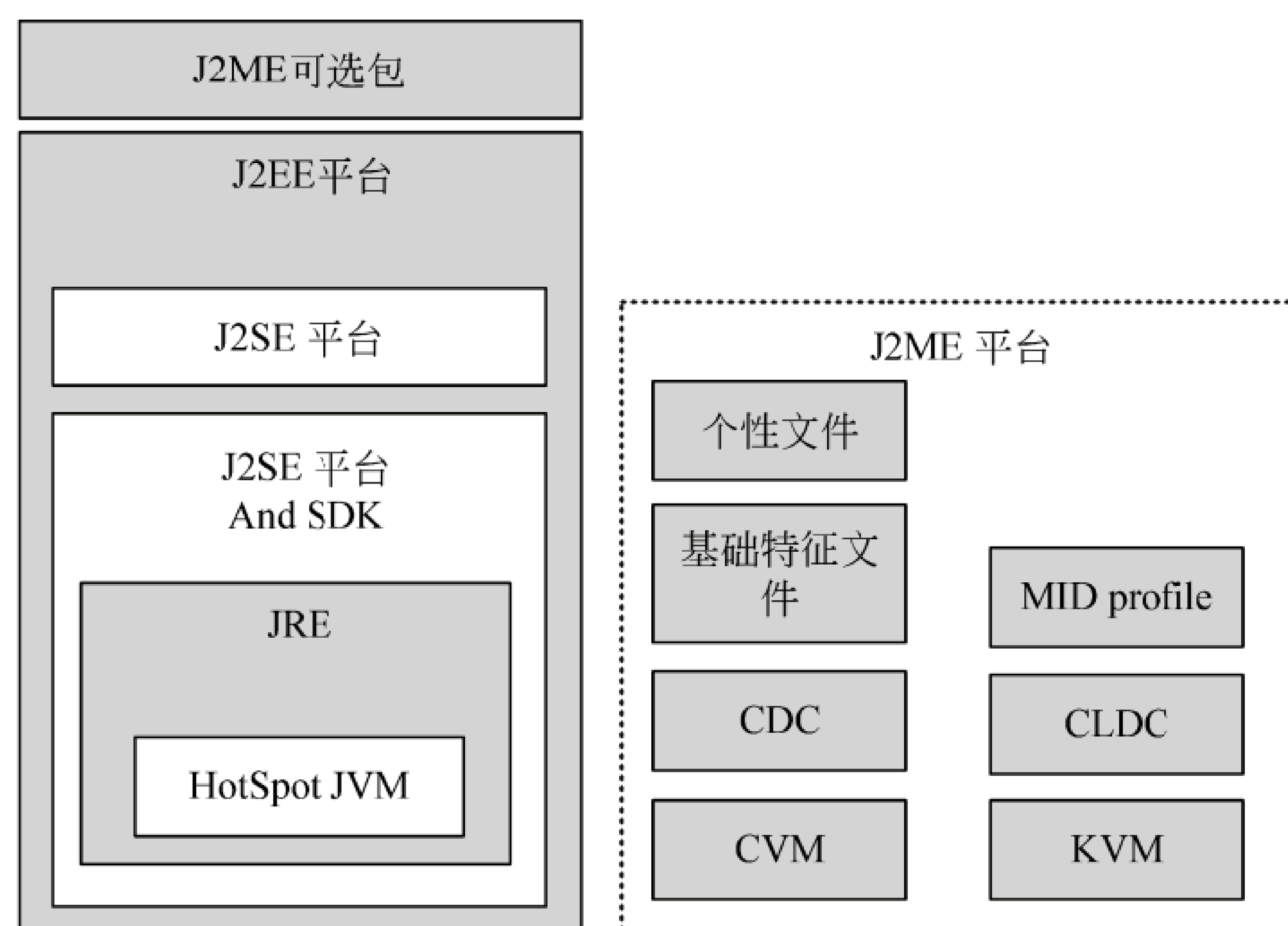


图 10-3　Java 2 组织结构

2．运行环境和参考实现

Java 运行环境（Java Runtime Environment，JRE）是 J2SE 平台的一部分，而 J2SE 本身又是 J2EE 平台的一个子集。JRE 包括运行时刻、核心库和浏览器插件。Sun 公司的 JRE 1.4 参考实现基于 HotSpot 运行时刻和提供对 JIT 编译的二进制代码进行在线再优化的 HotSpot JIT 编译器。单独的 HotSpot 编译器也有对应客户端和服务器端环境的版本。它们的区别在于根据内存占用历史信息、启动时刻、吞吐量和延时等不同而折中并对目标过程进行优化。Java SDK 1.4 在包含 Java 编译器、调试器、平台调试体系结构 API（JPDA）和用于生成文档（javadoc）的工具的同时，也包含了 JRE 1.4。图 10-4 给出了 J2SE 平台 1.4 版本的主要结构。

J2EE 体系结构概况通过使用专有的构件模型来区分了 J2EE 支持的范围。JavaBean 和它的核心技术可以被用在图中几乎所有的层次。此外，请注意图中的箭头表示了控制流的典型情况，当然，并不完全。数据流一般也沿着同样的路径，但在两个方向都存在。一个底层的用于支持 J2EE 所有部分的系统是通过 JNDI（Java Naming and Directory Interfoce，Java 的命名与目录接口）访问的命名和目录的基础结构。另一个集成平台是通过 JMS（Java Message Service，Java 消息服务）可访问的消息基础结构。在 EJB 容器的消息驱动构件的帮助下，消息在到来的时候可以触发处理过程。消息和命名/目录是两个重要的集成的层次服务，但也有一些其他的部分，比如事务协作和安全服务。



图 10-4 Java 2 平台标准版 1.4 的组织结构（资料来源：java.sun.com）

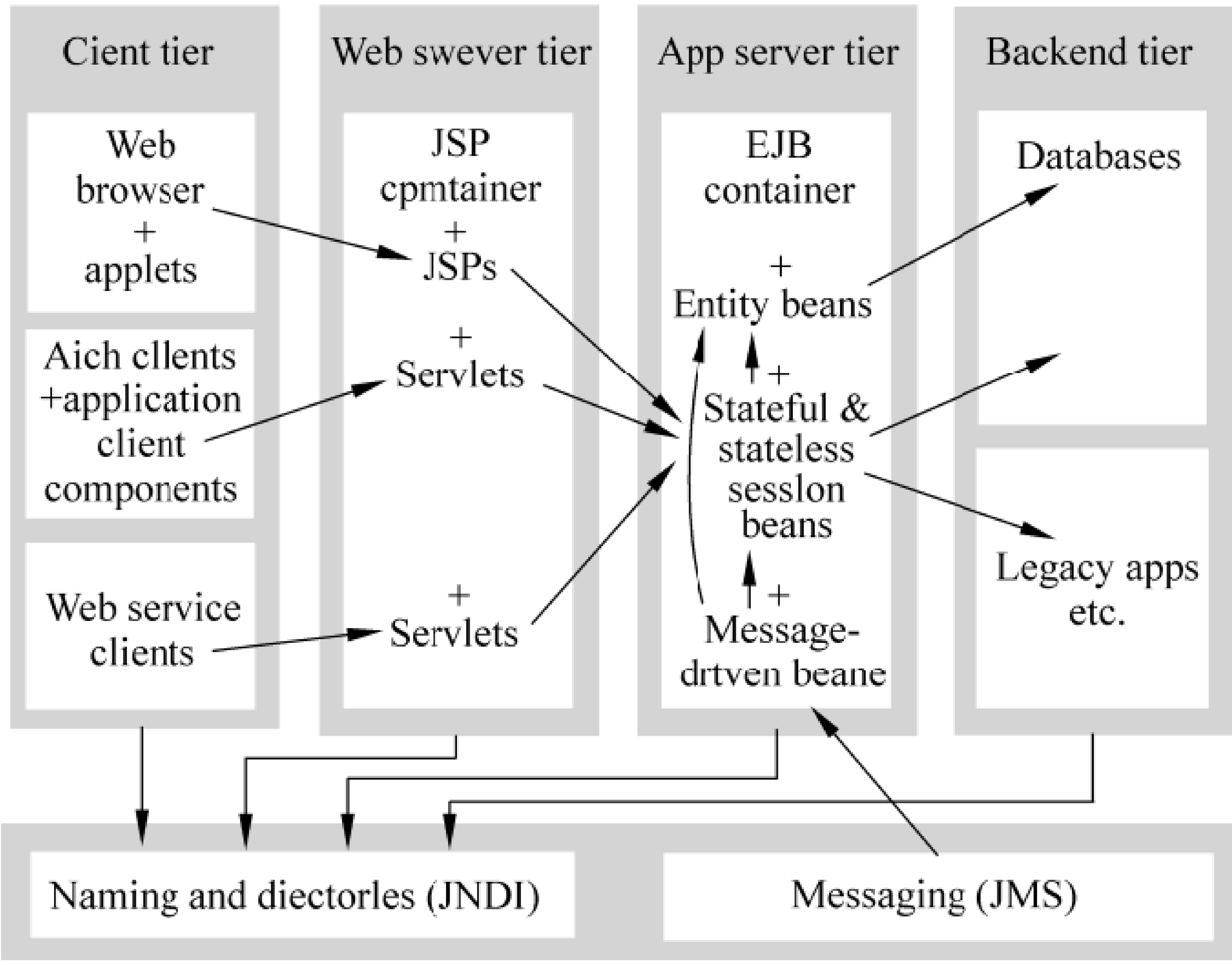


图 10-5 J2EE 体系结构概况

10.2.2　JavaBean

JavaBean 填补了部分空白，成为一种新的可行的产品——Java 构件，我们称之为 Bean（Sun，1996）。不幸的是，Java 中类和对象之间明确的区别并没有被贯彻到 JavaBean 中。尽管一个 Bean 的确是一个构件（一系列类和其他资源），但是它的那些定制好的连接的实例仍然被称做 Bean。这很让人迷惑。因此，我们用 Bean 来指代构件，用 Bean 实例来指代构件对象。“Bean 对象”将会带来困扰，因为一个 Bean 通常包含了许多 Java 对象。

Bean 模型主要包括以下几个方面。

（1）事件：Bean 可以声明它们的实例是潜在的事件源或者特定类型事件的监听者。一个组装工具能够把事件源和监听者连接起来。

（2）属性：Bean 通过成对的 getter 和 setter 方法暴露出一系列的属性。这些属性可以用来进行定制或者编程。属性的变化可以触发事件，也可以被事件强制修改。一个受限的属性只有在修改不被禁止的情况下才可以被修改。

（3）自检：一个组装工具能够检查一个 Bean，发现这个 Bean 的属性、事件，以及所支持的方法。

（4）定制：使用组装工具，一个 Bean 实例能够通过设置它的各种属性来完成定制。

（5）持久化：定制好的、已经连接的 Bean 实例需要进行保存，以便在应用程序使用它的时候重新装载。

10.2.3　基本的 Java 服务

经过这些年，Java 已经添加了许多标准服务。这一节我们将看到反射、对象序列化，以及 Java 本地接口。

1．反射

Java 的核心反射服务是一个集合体，包括原始 Java 语言的特性、一套支持类（在 JDK 1.1 中引入），以及支持类文字的语言特性。反射服务受到活动安全策略的约束，它允许我们：

（1）检查类和接口，包括它们的属性域和方法。

（2）构建新的类实例和新数组。

（3）对象和类的属性域的访问和修改。

（4）数组元素的访问和修改。

（5）对象和类的方法调用。

由此，反射服务涵盖了 Java 语言的所有特性。Java 语言级的访问控制机制，比如域的私有性，被大大增强了（无限制的访问对于实现可信任的底层服务，例如便携式调试器，是很有用的。为了实现这些无限制的访问，Java 平台的调试器体系结构提供了一

个特别的接口——JPDA（Java Platform Debugger Architecture）。为了进行反射操作，反射服务引入了包 java.lang.reflect。

类 Field，Method，以及 Constructor 提供了关于属性域、方法和构造器的反射信息，这些信息由它们描述，并对这个域、方法和构造器进行类型安全的使用。这三个类都是最终的，没有公有的构造器。它们三个都实现了接口 Member，这使得我们可以弄清楚成员如何被调用，确定成员的改动及该成员属于哪个类或者接口。

2．对象序列化

在 JDK 1.0.2 以前，Java 都不支持把对象序列化至字节流——仅仅支持基本类型。如果一个应用想把整个对象网写到输出流，它需要使用特别的编码方案来遍历和序列化对象自身。Java 对象序列化服务解决了这个问题，它通过定义一个标准的连续编码方案来达到目标，同时提供编码和解码（“序列化”和“反序列化”）对象网的机制。

一个对象能够被序列化，它必须实现接口 java.io.Serializable。另外，所有不应该被序列化的域需要用暂时修饰符标记。这一点很重要，因为域可能指向巨大的计算机结构（比如缓存）或者固有绑定到当前 JVM 的值（比如打开文件的描述符）。对实现 Serializable 接口的对象而言，足够的信息被写入一个流，以使得反序列化能继续进行，即使使用不同（但是兼容）的类版本。通过实现方法 readObject 和 writeObject，可以进一步控制将哪些信息或者添加更多的信息写入流。如果这些方法没有实现，所有指向可序列化对象的非暂时域将自动被序列化。指向对象的共享引用被保存起来。

3．Java 本地接口

Java 本地接口（Java Native Interface，JNI）为每一个平台规定了本地调用方式，在 Java 虚拟机之外我们可以调用本地代码。JNI 还规定了这些外部代码如何按照传递过去的引用来访问 Java 对象。这包括了调用 Java 方法的可能性。JNI 并没有规定 Java 二进制对象模型——也就是说，它没有规定在一个特定的 Java 虚拟机中属性如何被访问及方法如何被调用。同一平台上不同 Java 虚拟机之间的互操作仍然是一个未解决的问题，比如实时编译器这样的边界服务。JNI 允许本地方法：

（1）创建、检查和更新一个 Java 对象。

（2）调用 Java 方法。

（3）捕捉和抛出异常。

（4）装载类，获得类的信息。

（5）进行运行时刻类型检查。

4．Java AWT 和 JFC/Swing

Java 抽象窗口工具包（Abstract Windowing Toolkit，AWT）和 Java 基础类（Java Foundation Classes，JFC）提供了一个图形用户接口，这对于任何 Java 开发都具有重要意义。

基于委托的事件模型——这也许是在 JDK 1.1 中最富有戏剧性的改变。以前的事件

模型基于从构件类的继承及事件管理方法的重载。“基于委托”有一点用词不当，因为它沿用了 COM 中的术语委托。JDK 1.1 事实上提供了一个基于转发的事件模型。对象连接和组装被用来更好地实现继承。

- 数据传输和剪贴板支持：就像 COM 通用数据传输服务，AWT 定义了可传输数据项的概念。Internet MIME（多用途 Internet 邮件扩展）类型被用来和非 Java 的应用程序相互作用。Java 应用程序之间的数据传输也可以直接使用 Java 类。
- 拖放：支持在 Java 和非 Java 应用程序之间进行拖放（通过连接底层系统的拖放协议，比如 Windows 中的 OLE）。
- Java 2D：新的 2D 图形和图片类。Java2D 包含了线、文本和图片。支持的属性包括图片合成、Alpha 合成（透明化）、精确颜色空间定义和转化，以及面向显示的图像操作。
- 打印：打印模型比较简单。非显式地处理打印的图形构件将使用它们的屏幕透视方法被打印出来。因此，对于简单的构件，打印可以随时进行。然而，打印模型并不处理打印那些需要在多个页面分布的嵌入内容（ActiveX 打印模型对此提供了支持，它和容器合作，允许嵌入的内容跨业若干页进行打印。这个是一个复杂的模型，然而只有很少的 ActiveX 容器真正实现了这个高级打印模型）。
- 可访问性：允许所谓的辅助技术来和 JFC 及 AWT 构件交互的接口。辅助技术包括屏幕读取器、屏幕放大器和语音识别。
- 国际化：以 Unicode 2.1 字符编码为基础，提供了对文本、数字、日期、货币，以及用户自定义对象和当地习惯相适应的支持，使用语言和区域标志符来识别正确的格式。

10.2.4 各种构件——Applet，Servlet，Bean 和 Enterprise Bean

在 Java 领域中定义了 5 种不同的构件模型，而且将来可能会出现更多。这其中不仅包括了 Applet 和 JavaBean 模型（J2SE 的一部分），还有 Enterprise JavaBean，servlet 和应用程序客户端构件（J2EE 的一部分）。本节将对这些不同的构件模型进行简要地描述。

servlet/JSP 和 EJB 是 J2EE 服务器端模型的两个关键元素，在下面的论述中将对它们进行更详细的介绍。在 J2EE 中，所有的构件将被打包成 JAR 文件，这样 J2EE 应用就可以将这些 JAR 文件包含进来。J2EE 中的构件都有一个很重要的方面，就是它们都支持部署描述符（Deployment Descriptors）。部署描述符是一个 XML 文件，和相应的构件一起打包，用来描述这个构件应该怎样进行部署。部署是指根据实际的部署语境将一个构件进行准备的动作，这一步骤可以是，也经常是从安装软件的概念中分离出来的。部署描述符的详细内容依赖于特定的构件模型。例如，一个 EJB 实体 Bean 的描述符，就要有容器管理的持久性的描述，以及对 EJB 实体 Bean 中的属性到数据库中表的映射的详细描述。

Applet 是第一个 Java 构件模型，用于轻量级的可下载的构件，以增强网站在浏览器中的视觉效果。最初的 Applet 安全模型非常严格，以致 Applet 不比 eye candy 发送得多。eye candy 是指在通过别的技术捕捉一个区域，这些技术包括 GIF 动画、Macromedia Shockwave 和 Flash 技术、JScript、对 HTML 的增强（包括 DHTML（动态 HTML）的引入）等。为了充分利用浏览器端技术，绝大多数的基于 J2EE 的应用均使用 servlet 和 JSP 来通过脚本生成 HTML 页面，而不是通过下载 Applet 来实现。

第二个 Java 构件模型是 JavaBean。它主要用于支持基于连接的程序，比如同时用在客户端和服务器端的程序。从历史上来说，JavaBean 在客户端占比重较大的应用中使用得更广泛，而在服务器端有时候还会被 EJB 所替代。但这种观点从技术角度上来说是错误的：EJB 远远不只是它的名字看上去的那样，和 JavaBean 的相似之处也很少。当建造一个明确支持可视化应用设计模式的客户端应用时，JavaBean 依然是有用的（就像 J2SE 1.3 中所述，一个 Bean 也可以是一个 Applet。但无论如何，这种支持是有限的——BeanApplet 所接受的总是空的 Applet 语境和存根）。

EJB 是 Java 的第三个构件模型。它使用容器集成服务，用以支持 EJB（构件）使用声明属性和部署描述符的方式来请求服务。在最新的修订版中，JavaBean 也加入了容器模型，但它的容器模型与 EJB 容器有很大的不同。前者仅仅是一种容纳的机制，而后者则是一种声明型构造驱动的模式。因为 JavaBean 不需要在设计时之外与用户交互，所以也可以使用 JavaBean 来构造更复杂的 EJB。（JavaBean 和 EJB 与.NET 框架中的构件类及被服务的构件类大致对应。）

第四个 Java 构件模型是 servlet。它跟 Applet 相似，但属于服务器端构件模型，而且是通常由 Web 服务器进程进行实例化的轻量级构件，Web 页面就是一个典型的例子。Java Server Page（JSP）是一种与之匹配的技术，能够声明式地定义要生成的页面，然后 JSP 会被编译成 servlet。

J2EE 引入的第五种 Java 构件模型为应用客户端构件。它本质上是在客户端的不受限制的 Java 程序。一个客户端构件通过用命名语境的 JNDI 企业来访问 J2EE 服务器中的环境属性、EJB 和各种资源。这些资源可以包括对 E-mail（通过 JavaMail）或数据库（通过 JDBC）的访问。

对 Java 构件模型不同种类的支持是不同的需求的反映。但无论如何，要在这些不同的领域实际地建立构件市场，还需要在基于领域的概念等方面建立更深入的标准化。现在，只有很少的 EJB 构件在需要开发它们的系统之外的其他系统中使用。

10.2.5 高级 Java 服务

本节将介绍 Java 如何在企业级范围支持分布式计算。实际上 Java 中有 4 种模式支持分布式计算——RMI，RMI over IIOP，CORBA 和 EJB 容器（本身是建立在 RMI 或 RMI over IIOP 上的）。EJB 在前面已经论述了。本章将论述其他的几种，以及一些最重

要的支持分布式应用的服务。

1．分布式对象模型和 RMI

分布式计算主要由对象序列化服务和远程方法调用（Remote Method Invocation，RMI）服务支持。这两种服务都是在 JDK 1.1 引入的。下面的章节介绍 RMI 和 RMI over IIOP，它们有细微的差别。

一个分布式对象的句柄为一个接口类型的引用——它不能指向一个 remote 对象类及其超类。能够远程访问的接口必须直接或间接从 java.rmi.Remote 继承下来。一个远程操作可以由于网络或远程硬件故障而失败。remote 接口的所有方法都要声明检查 java.rmi.Remote Exception 异常。将参数传给远程操作很有意思，如果一个参数是 remote 接口类型，那么就是按引用来传，其他类型则按值来传——这就意味着，参数将在调用端序列化，在调用 remote 接口方法的时候反序列化。Java 对象不需要序列化。如果企图传一个无法序列化的对象将会抛出运行时刻异常。如果语言规定使用 Java RMI，那么编译器可以静态地规定只有可序列化对象可以通过值来传送，而且所有的方法都声明 Remote Exception 异常。

Java 分布式模型扩展了垃圾回收。它将所有有远程引用指向它们的对象都记录下来，这样就可以支持分布式的垃圾回收了。回收器是基于 Network Object（Birrel，1993）的工作的。分布式垃圾回收是 RMI 比当今其他模型出色的一点。唯一的另外一种基于 Network Object 的和有租用引用思想的就是 CLI remoting，但这种方法的提出比 Java RMI 晚了四年。

2．Java 和 CORBA

一个 OMG IDL 到 Java 的绑定和 OMG 最先给出的 Java 到 OMG IDL 的绑定，是 1998 年在 CORBA 2.2 中定义的。将 CORBA 包含在 Java 项目的一个重要原因是能用 IIOP 和非 Java 系统通信。为了能访问 CORBA 服务，通常使用 Java 规范定义的接口会更方便，这些接口能够映射到兼容 CORBA 或其他服务。

3．企业级服务接口

J2EE 的重要部分就是一些适合于企业级服务的接口。这些服务接口也可以通过 CORBA 来建立。然而，Java-CORBA 的集成必然会引起一些冲突。与此相反，在本节讨论的以 Java 为中心的接口，从客户和实现者的角度来减小这些不足。

1）Java 命名和目录接口（JNDI）

在计算系统中的一个全局性问题就是通过名字或属性来定位服务的问题。命名服务针对前一个问题，目录服务针对后一个。命名服务的例子包括 Internet 域名服务（DNS）、RMI 注册表和 CORBA 命名服务。目录服务的例子包括兼容 LDAP 的目录系统，比如 Novell 的 eDirectory、微软的 Active Directory 和开放源码的 OpenLDAP（www.openldap.org）。

JNDI 为命名服务（javax.naming）和目录服务（javax.naming.directory）提供了统一

的 API。Context 这个最普遍使用的接口使命名语境对 lookup 方法有效，使用这个方法就可以根据名字来定位对象了。一个命名语境也可以用来对绑定在某个语境中的名字进行列表，或者是解除一个绑定，或是创建或删除一个子语境。

EJB 的 Bean 的一个重要的命名语境就是 EJB 容器提供的环境命名语境(Environment Narning Context，ENC)。通过它可以访问环境属性、其他 Bean 和资源。接口 DirContext 扩展 Context 来提供目录功能，包括检查和更新与目录上列出的对象相关联的属性，以及通过值来搜索一个目录语境。因为 DirContext 是从 Context 继承而来的，所以一个目录语境也是命名语境。绝大部分的语境是通过对其他语境的递归查找而找到的。而查找的起始点就是通过初始化 InitialContext 这个类得到的。

JNDI 也定义了一个事件 API（javax.naming.event）、一个支持 LDAP v3 的超过 DirContext 的功能，以及一个能够使命名和目录服务的提供者与 JNDI 连接起来的服务提供者接口（javax.naming.spi）。事件机制用来为改变通知进行注册。J2SE 1.4 内置了 4 种服务提供者——CORBA 命名、DNS、LDAP 和 RMI。

2）Java 消息服务（JMS）

异步消息是将实例的操作和覆盖的组装模型通过消息进行通信。基于事务的消息队列建立的可靠性级别，正常情况下需要基于同步调用的模型。灵活的消息路由、广播和过滤增强了灵活性。JMS 是 Java 对消息系统的访问机制，但它本身并不实现消息。

JMS 支持点对点分发的消息队列，也支持多个目标订阅的消息主题。当消息发布给一个主题的适合，消息就会发送给所有那个主题的订阅者。JMS 支持各种消息类型（二进制、流、名-值表、序列化的对象和文本）。通过声明与 SQL 的 WHERE 相近的句段，可以建立消息的过滤器。

3）Java 数据库连接（JDBC）

JDBC 是根据流行的微软 ODBC（Open DataBase Connectivity，开放数据库连接）标准建立的一个通用的与数据库交互的方法。JDBC API 分成核心 API（在 java.sql 包和 J2SE 的一部分中）和 JDBC 可选包（在 javax.sql 包中，J2SE 可选但 J2EE 必须遵循）。JDBC 像 ODBC 一样，需要驱动程序将 JDBC API 映射到特定数据库的本地接口。

JDBC 驱动程序有 4 种。Type 1 和 Type 2 驱动通过 JNI 来使用本地代码（非 Java 代码）。Type 1 驱动使用具有通用接口的本地代码，而 Type 2 允许使用数据库特定的接口。最普遍的 Type 1 驱动就是 JDK 包含的 JDBC-ODBC 桥，它将 JDBC 调用映射为 ODBC 调用。因为 ODBC 是用字句的驱动模型来访问特定的数据库，因此这种方式相对会慢一些。Type 3 和 Type 4 驱动都是纯 Java 的。Type 3 通过网络协议和数据库网关来间接地访问数据库，Type 4 则是直接访问数据库。驱动程序的选择不会影响客户的代码，因为 JDBC API 本身是不受驱动影响的。对于性能来说，Type 4 通常是最好的，接下来是 Type 2，再接下来是 Type 1，最后是 Type 3。

4）Java 事务 API 和服务（JTA，JTS）

事务管理常常是由 EJB 容器来委派的，但有些情况还需要显式的事务管理。CORBA 对象事务服务（OTS）或者它的 Java 实现（Java Transaction Service，JTS）可以用于这个目的。然而，EJB 中有一个更为简单的接口，称为 Java 事务 API。它是服务器/容器的实现使用的低级 XA 接口（X/Open 事务 API 接口标准）和 EJB Bean 的实现可以访问的高级客户端接口。

在 JTS（或 OTS）中，需要显式地和仔细地使用事务中的资源，这样，这个显式的对事务的划分会形成一个边界。高级 JTA（Java Transaction API）接口，这个容易出错的功能是由 EJB 容器来完成的。然而，由于资源由长事务所掌握，显式的事务管理仍然是很容易出错的，并且将引起不一致或效率低下。

4．J2EE 连接器架构（JCA）

JCA 标准化连接是由 J2EE 1.3 首先提出的，它位于 J2EE 应用服务器和企业信息系统（Enterprise Information System，EIS）之间，比如数据库管理、企业资源规划（ERP）、企业资产管理（Enterprise Asset Management，EAM）和客户关系管理（CRM）系统。不是用 Java 开发的企业应用或者在 J2EE 框架内的应用都可以通过 JCA 连接。JCA 是在 javax.resource 包和它的子包（cci，spi 和 spi.security）中定义的（JCA 的形式也用于 Java cryptography API 的缩写）。

5．Java 和 XML

Sun 是 XML 的一个早期提倡者。然而，最初 Java 对 XML 的支持只是限定在定义能够处理 XML 文档的接口，能够表示 XML 文档（Document Object Module，DOM）和 XML 流（Sinple API for XML，SAX）的模型。而更多对 XML 支持，包括对 XML Schema 和 Web 服务标准的支持已经作为预发布版本，在 2002 年初加了进来。

10.2.6　Java 和 Web 服务——SunONE

SunONE（Sun 开放网络环境）是 J2EE 的扩展，它可以通过特殊的 servlet 来处理 Web 服务协议。SunONE 也包含了以前由 iPlanet 策划的 J2EE 服务器产品。（注意：Netscape 公司和 Sun 公司组建的 iPlanet 联盟在 2002 年年初已经结束了，并把 iPlanet 开发的产品留给了 Sun 公司。在 2002 年年初，iPlanet 在 J2EE 的市场份额中占 7%，排名在它之前的是 IBM 公司的 WebSphere 和 BEA 公司的 WebLogic（各占 34%），紧随其后的是 Oracle 公司（占 6%）。随着在 2002 年年初 JavaWeb 服务开发包（JavaWSDP）可用版的发布，Sun 公司对 SOAP、WSDL、UDDI 都提供了支持。JavaWDSP 包括了 Java 为 XML 消息处理（Java API for XML Messaging，JAXM）、XML 处理（Java API for XML Processing，JAXP）、XML 注册（JAXR）和基于 XML 的 RPC（JAX-RPC）提供支持的 API。另外它还包括 JSP 标准标签库（Java Server Pages Standard Tag Library，JSTL）、Ant 创建工具、Java WSDP 登记服务器、网络应用工具，以及 Apache Tomcat 网络服务器容器。

10.3 Microsoft 的方式

从某种意义上讲，微软选择的是最简单的路线。它没有提出一整套标准，并期望依此改变自己的系统。相反，它不断地对已有的应用和平台基础进行再工程。构件技术是渐进引入的，这就可以获益于以前的成功技术，例如，Visual Basic 控件（VBX，一种不是面向对象的构件！），对象链接和嵌入（OLE），OLE 数据库连接（ODBC），ActiveX，微软事务服务器技术（MTS），以及主动式服务器端页面技术（ASP）。

在技术标准的领域里，微软的主要兴趣放在 Internet 标准（Internet Engineering Task Force，IETF）和 Web 标准（W3C）上。最近，它的.NET 规范的一部分被 ECMA（European Computer Manufacturers Association）组织采纳。ECMA 是欧洲的一家标准化组织，它被视为通向 ISO 的捷径（参见 ECMA，2001a，2001b）。微软并未试图让自己的方法与 OMG 或 Java 标准保持一致。尽管 Java 曾在微软的战略中扮演过一段重要角色，它目前的地位却已不高，而仅仅是为了继续支持一项较老的 Visual J++产品。这其中有一部分是解决 Sun 公司和微软公司争讼所带来的后果。而且，微软还重点瞄准使用 Visual J++6.0 的用户，靠着 Visual J# .NET 的名称，推出一个向.NET 迁移的工具。

作为.NET 计划的一部分，微软正在推动语言无关性，把它作为 CLR 的一条主要原则，并构造了一种新的语言 C#。C#吸纳了 Java 的很多成功特性，另一方面，它又新增若干独有的特征（例如值类型），且不支持某些关键的 Java 特征（例如内部类）。C#虽然定位于 CLR 的模型语言，但它与若干其他语言占有同等地位，包括被全面革新过的 Visual Basic .NET，Managed C++（对 C++的扩展，它与 ANSI 兼容），以及许多被其他供应商或组织支持的语言。

在依赖语境的组装方面，微软、OMG 和 Sun 这些公司技术之间的螺旋演进很有意思。依赖环境的组装最先在 COM 套间模型被粗略描述，又在微软事务服务器（Microsoft Transaction Server，MTS）中被丰富，在被 Enterprise JavaBean 采纳和改进的同时它又在 COM+技术中独立地发展，后来被 CORBA 构件模型（CCM）采纳和改善，最后，它变为 CLR 中的一项可扩展的和开放的机制，与此同时，EJB 2.0 的发展超越了意欲成为各项技术超集的 CCM，这意味着 CCM 规范进入了“维护阶段”。

COM 可能在未来多年内仍很重要，而且，CLR 与它的互操作能力格外强。鉴于此，下面对微软方法的讨论就以 COM 介绍开始。COM+在 COM 基础上新增了服务，CLR 的首次发布是使用 COM 互操作来提供 COM+服务的，因此，许多服务并非冗余。

10.3.1 第一个基础关联模型——COM

COM 是微软平台上所有构件的基石，微软还将之在 Macintosh 系统实现。在其他的许多平台，COM 也被诸如 Software AG、惠普这样的第三方厂商实现。即便如此，可以

说，COM 从未在微软的 Windows 平台之外赢得更多支持。但是，COM 的基本理念却有着相当的影响力。

COM 所定义的一个基础实体是接口。在二进制层面上，一个接口被表示为指向一个接口结点的指针。而一个接口结点唯一被指定的部分是置于其内部第一个域的另一个指针，这个指针指向一个过程变量表（或者说，函数指针表）。这些表源自 C++等语言用来实现虚函数（方法）的表，因此，也被称做 vtable。图 10-6 所示为二进制层面的一个 COM 接口。

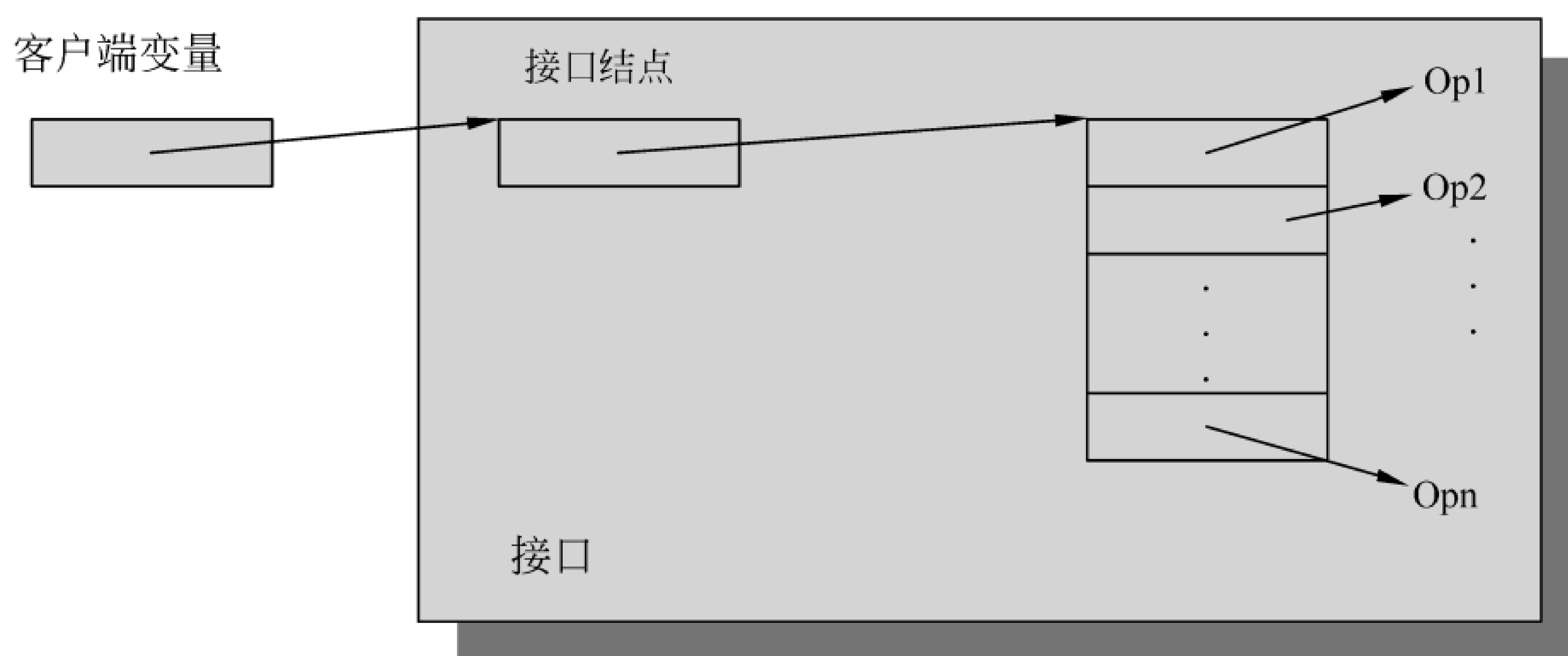


图 10-6　COM 接口的二进制表示

表示 COM 对象的通常方法是将其画成带有插口的盒子。由于每个 COM 对象都有 IUnknown 接口（它标志着 COM 对象），通常把 IUnknown 接口置于 COM 对象图的顶端。图 10-7 所示为 COM 对象图，这里是一个 ActiveX 文档对象。

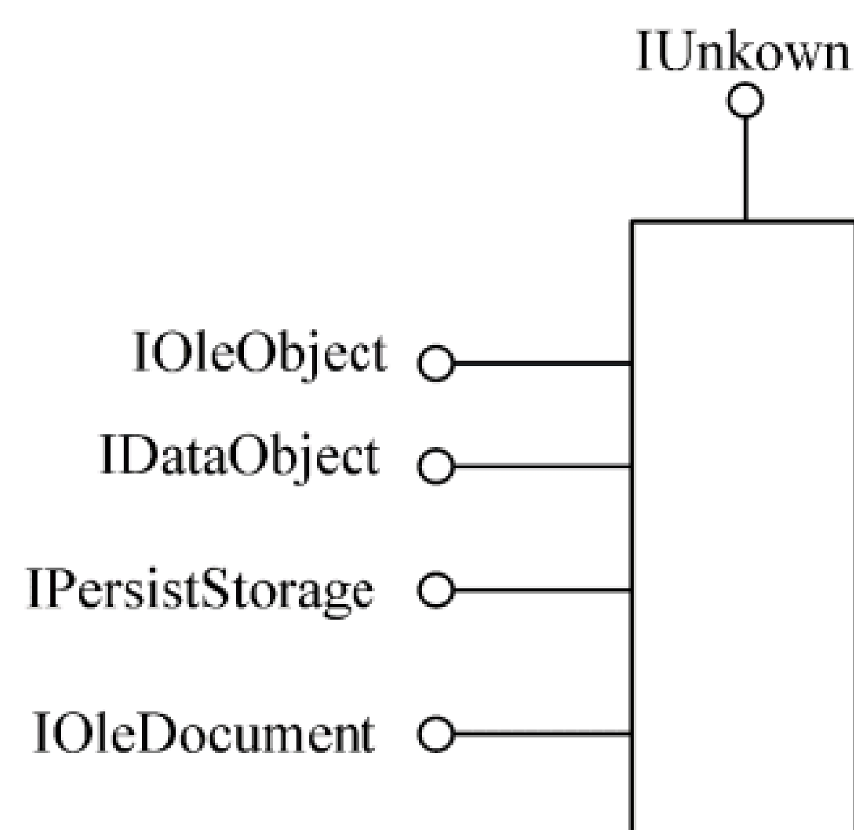


图 10-7　COM 对象描述

回到 IUnknown 接口。它的“真实”名字当然是它的 IID，即 00000000-0000-0000- C000-000000000046。但为了方便，所有接口也有一个可读名。根据习惯，可读接口名以字母 I 开头。与 IID 不同，可读名并不保证是唯一的。因此，编程中的接口引用均使用 IID。

IUnknown 接口的首要用途是在最抽象的情况下标志 COM 对象，此时 COM 对象没有任何特殊功能。因此，IUnknown 接口的引用可被用来和 ANY 类型的引用或面向对象语言的 Object 引用比较。从某种意义上，IUnknown 用词不当。它并不是一个未知的接口，相反，它是唯一能被保证永远存在的接口。对于一个没有别的已知接口的 COM 对象，它被获知的唯一途径就是通过 IUnknown 接口的引用。

IUnknown 接口只提供对任何 COM 接口都必须的三个强制性方法。第一个强制性方

法是 QueryInterface，前面已提及。另两个强制性方法名为 AddRef 和 Release。结合关于何时调用的规则，这两个强制性方法被用来控制对象的生命周期。后面会有更多解释。使用类似 COM IDL 的表示法，IUnknown 可被定义为：

```
[uuid (00000000-0000-0000-C000-000000000046) ]
interface IUnknown {
HRESULT QueryInterface (
[in] const IID iid, [out, iid_is(iid)] IUnknown iid);
unsigned long AddRef ();
unsigned long Release();
}
```

类型 HRESULT 被大多数 COM 接口的方法用来表示调用的成功或失败。QueryInterface 则用它表明查询的接口是否被支持。如果接口属于一个远程对象，HRESULT 也可能表示网络错误。

每个 COM 对象都会进行引用计数，或者是对整个对象体，或者是对每个单独的接口结点。引用计数变量被共享使用的情况下，COM 对象不能释放接口结点，即使这个结点已经没有引用。一般来说，这样做没有问题。而共享一个引用计数变量也是通常的做法。可是，某些情况下，接口结点会占据很多资源，例如当它们保留着一个大缓冲结构时。对于这类接口结点，可以使用独立的引用计数变量，以便结点可以尽早释放。这种根据需要创建和删除接口结点的技术有时被称做“快速装卸接口（Tear-Off Interface）”。

当对象或结点被创建，其第一个引用被传出之前，引用计数变量会初始化为 1。之后，每逢一个引用复制被创建，计数值必须增加（AddRef）；每逢一个引用被丢弃，计数值必须减少（Release）。在引用值变成 0 的瞬间 COM 对象无法被访问，因此它就该自行销毁。销毁工作的一部分是通过调用 Release 方法释放对其他对象的引用。其后果是，被正待销毁对象引用的所有对象都会被递归销毁。最终，被销毁的对象释放它所占有的内存空间。

10.3.2 COM 对象重用

COM 不支持任何形式的实现继承。注意，COM 没有定义或考虑单独的构件从内部如何去实现。构件可以由使用了实现继承的类组成。无论何种情况，缺少实现继承并不意味着缺少对重用的支持。为实现对象重用，COM 支持两种形式的对象组装：包含（Containment）和聚集（Aggregation）。

包含就是一种简单的对象组装技术，其含义是一个对象拥有指向另一个对象的唯一引用。从概念上来说，前者（称做外部对象）“包含”后者（称做内部对象）。外部对象只是把请求转发给内部对象。所谓转发，就是调用内部对象的方法，以实现对某个外部对象方法的调用。

包含能重用内含于其他构件的实现。特别是，对于使用外部对象的客户程序，包含是完全透明的。调用接口函数的客户无法辨别调用是由提供接口的对象处理，还是被转发给另一个对象处理。如果包含层次较深，或者被转发的方法本身相对简单，包含会存在性能上的问题，因此 COM 定义第二类重用形式，即聚集。聚集的基本思想很简单，直接把内部对象的接口引用传给外部对象的客户，而不再转发请求。对此接口的调用将直接到达内部对象，从而省去转发的代价。当然，只有在外部对象不希望截取调用以执行诸如过滤等额外处理时聚集。还有，保持透明性是很重要的，因为外部对象的客户无法辨别哪个特定接口是从内部对象聚集而来的。

10.3.3　接口和多态

COM 接口可通过（单）接口继承从其他 COM 接口中派生。实际上，所有 COM 接口都直接或间接地继承了 Iunknown，它是接口体系中的公共基类型。除了 Iunknown 外，只有 Idispatch 和 Ipersist 这两种重要的基接口被公共继承。COM 中接口继承为什么如此鲜为使用呢？

令人吃惊的是，COM 的接口继承与其支持的多态无关。例如，假定客户持有一个接口，比方说 IDispatch 的引用。实际上，客户引用的接口可以是 IDispatch 的任何子类型。换句话说，函数表可以包含 IDispatch 所需之外的方法。但重要的是，客户无法发现这一点。如果客户想要更特殊的接口，必须使用 QueryInterface。这样就能保证获得更多的方法，至于返回的接口结点实际上是否就是 QueryInterface 被调用发出的那个结点，对客户来说没有关系。

接口和版本化。一旦公布，COM 接口和它的规范不允许以任何形式改变。这种避免的方法既解决了语法问题，也解决了语义上的脆弱基类的问题。换言之，COM 中的 IID 可用于标志接口中的版本。因为接口总是通过 IID 被请求的，系统中的所有参与者都对接口的版本达成一致。CORBA 讨论中所提及的传递性版本冲突问题在 COM 中不会发生。

构件可以选择实现接口的多个版本，只不过处理方式就像处理任何别的不同接口一样。使用这种策略，基于 COM 的系统能并发支持旧接口和新接口，同时允许渐进的迁移。在某些系统中，由单个对象实现的多个接口被合并成单个类的命名空间，类似上述的策略实现起来就变得困难，或至少不自然。对于建立在传统对象模型（像 Java 或 CORBA）之上的方法，这会给二进制兼容性带来问题。CLR 避免此问题的方法是，在相同的类实现的不同接口上，允许分别实现具有相同名字和签名的方法。除此之外的其他方面，CLR 还是基于传统对象模型。

10.3.4　COM 对象的创建和 COM 库

创建对象的最简单方法是调用 CoCreateInstance（所有 COM 库的过程名以 Co 起

头，它代表 COM)。此函数需要一个 CLSID 和一个 IID，然后创建指定类（CLSID）的新实例，并返回所请求类型（IID）的接口。如果 COM 无法定位或启动能实现所请求 CLSID 的服务器，或者指定的类不支持所请求的接口，就会返回错误提示。

创建 COM 类的实例对象时，COM 需要把给定的 CLSID 映射为包含所请求类的实际构件。为此目的，COM 支持系统注册器，它类似 CORBA 存储器。注册器指明哪些服务器是可用的，它们支持哪些类。服务器可以是进程内（inprocess）服务器、本地服务器和远程服务器这三种类型中的一种。进程内服务器支持存在于客户进程中的对象；本地服务器支持的对象位于客户所在的机器上，但在不同的进程内；远程服务器支持的对象位于不同的机器上。

CoCreateInstance 接受一个额外的参量，用于指定何种服务器是可接受的。CoCreateInstance 查询注册器以定位服务器。若服务器尚未被激活，就载入并启动它。对于进程内服务器，需要载入和链接动态链接库（DLL）。而对于本地服务器，独立的可执行文件会被载入。最后，对于远程机器，会联系远程机器上的服务控制管理器，以载入并启动该机器上的服务器（以中间件观点看，SCM 起着类似 CORBA ORB 的作用）。

COM 服务器具有定义好的结构，包含一个或多个类，对每个类它又实现一个工厂对象（在 COM 里，工厂对象被称做类工厂。这个名称可能让人误解，因为工厂创建的不是类，而是类的实例）。工厂是支持 IClassFactory 或 IClassFactory2 接口的对象，使用后一个接口意味着需要许可机制。COM 使用工厂的原因是，COM 对象不一定是简单的单体对象（single-object），因此其创建需要由其构件而非系统提供的服务来指定。

图 10-8 含两个 coclasses 的 COM 服务器，每个都有一个工厂。启动时，自注册服务器为每个类创建一个工厂对象，并将之注册到 COM。CoCreateInstance 使用工厂对象创建实例。为了提升性能，客户也可以使用 CoGetClassObject 获得对工厂的直接访问。在需要创建许多新对象时，这种做法较有用。很多时候，客户所要的不是具体的类，而是更一般的东西。例如，客户并不使用对应 Microsoft Word 的 CLSID，而是使用对应 rich text 的 CLSID。为了支持这种一般性 CLSID 和相应的配置，COM 允许一个类仿真另一个类。仿真配置保存在系统注册器里。例如，某个仿真项也许会指定类 Microsoft Word 仿真类 rich text。

10.3.5 从 COM 到分布式 COM（DCOM）

COM 透明地扩展 COM 的概念和服务。DCOM 中已存在客户端代理（Proxy）对象和服务器端桩（Stub）对象，它们只被用于支持进程间通信。DCOM 建立在这两者的基础上，在前面谈到远程服务器时已暗示过 DCOM 服务。

为支持跨进程或跨机器的透明通信，COM 在客户端创建代理对象，在服务器端创建桩对象。为了单个机器内进程间的通信，代理和桩需要实现的，仅仅是从简单数据类型到字节流和从字节流到简单数据类型的映射。因为发送和接收进程在同样的机器上执

行，所以不需要担心数据类型是如何表达的。而当接口引用被传递时，尽管仍在相同机器的不同进程间，情况也会变得稍微复杂些。

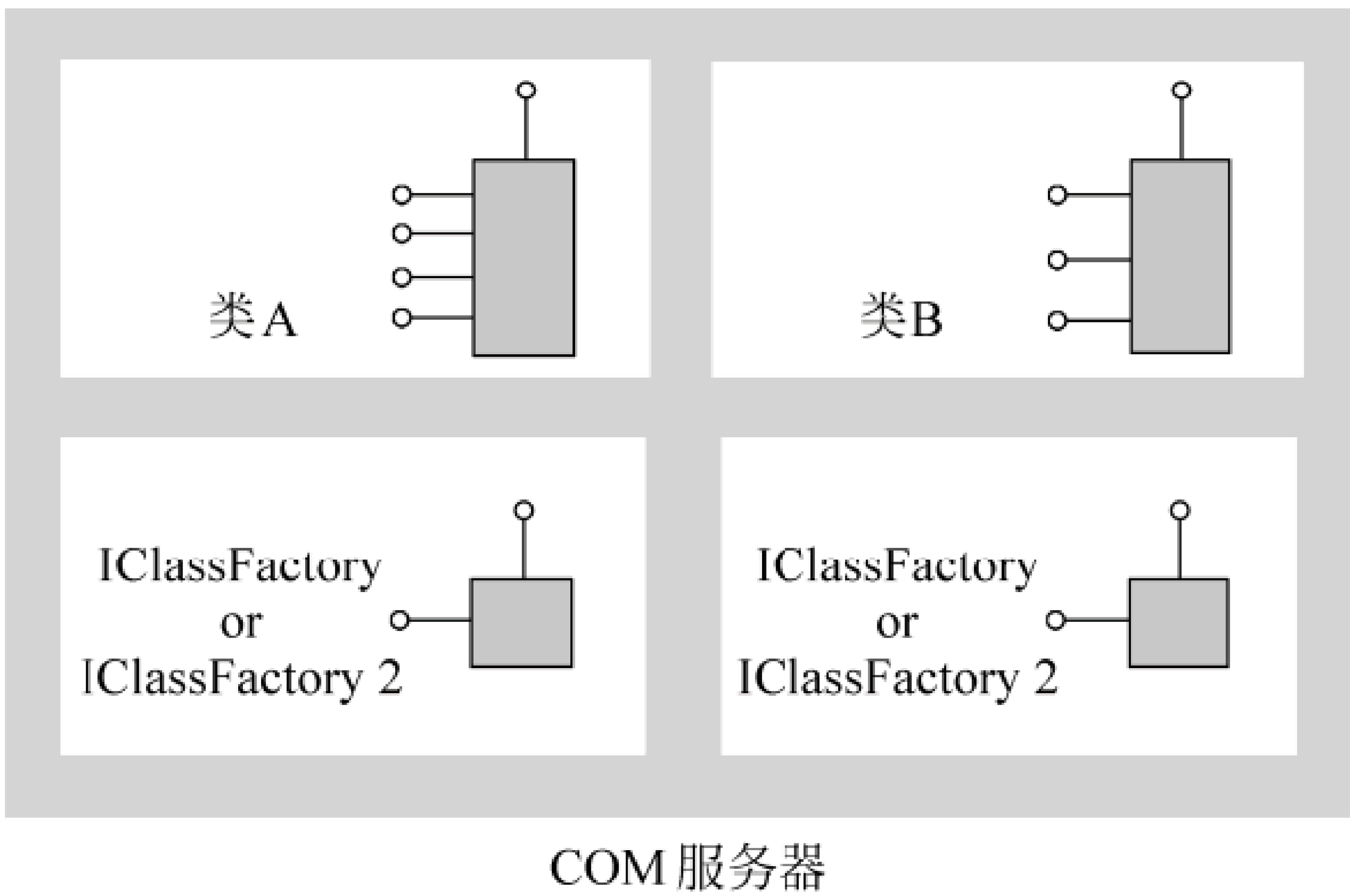


图 10-8　一个 COM 服务器支持两个各带工厂的 COM 类

跨进程传递的接口引用需要被映射为对象引用，它的意义在穿过进程时仍能维持不变。当接到对象引用时，COM 需要确定对应的代理对象存在于接收端。然后，COM 选择该代理的对应接口，并传送这个接口引用而非先前的那个接口引用。先前的引用会指向“错误”进程的接口。

图 10-9 显示了客户向对象 A 发出一个调用。被调用的方法只有一个参量，它引用对象 B 的一个接口。由于对象 A 位于另一进程，本地代理对象中转此调用。代理决定对象 B 的对象标志符（OID）和被传递接口的接口指针标志符（IPID）。OID 和 IPID 一起随着客户进程 ID 被传递给服务器进程的桩。桩使用 OID 定位对象 B 的本地代理，使用 IPID 定位具体的接口。接着，桩代表客户发出先前的调用，它将本地 B 代理的接口引用传给调用接受者——对象 A。

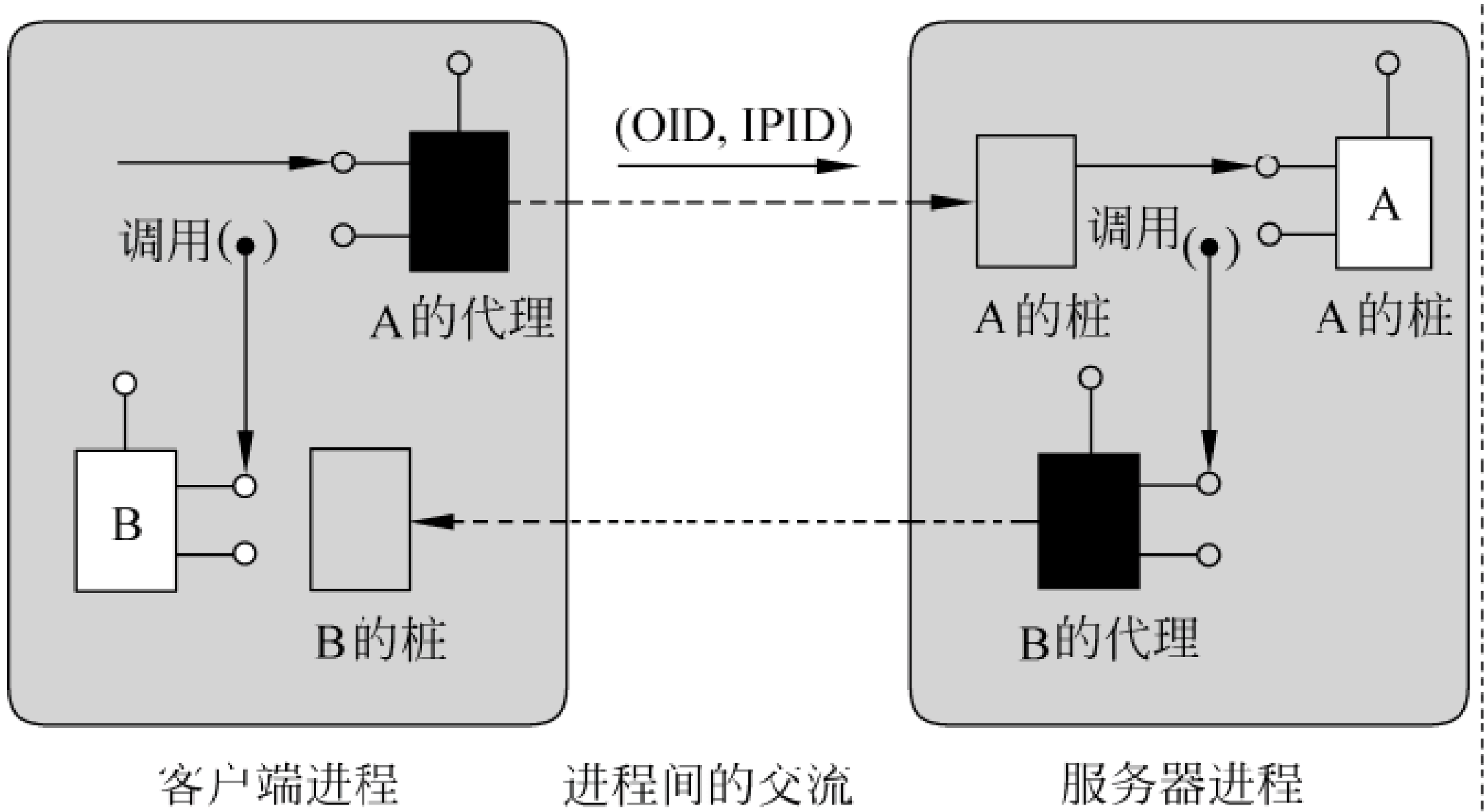


图 10-9　单机上进程间接口引用的编排与起源

DCOM 所用的方法相当近似。有两点不同，在不同机器上数据类型的表达可能是不一样的，并且对象引用需要包含比 OID 和 IPID 更多的信息。为了解决数据表达的差异，DCOM 将数据整理成平台无关的网络数据表达（NDR）形式。为了形成与机器无关的对象引用，DCOM 将 OID、IPID 和那些足以定位对象输出器（Object Exporter）的信息结合在一起。对象输出器是 DCOM 提供的对象，它知道如何绑定服务器公布的对象。每个对象输出器有唯一的标志符（OXID），它被包括在对象引用中。

10.3.6 复合文档和 OLE 对象

链接和嵌入是微软的复合文档标准。创造 OLE 的原本意图是为了将各以应用为中心的遗留程序融合成单一的以文档为中心的范型。也有可能创建只存在于 OLE 环境中的对象，ActiveX 就是最好的例子。然而，OLE 也继续用变化的 OLE 集成程序来支持独立的应用。这种实用化的一面使很多 OLE 技术非常复杂。然而，可以按平滑的路径进行技术迁移，以保护在开发和用户培训方面的投资，这样就可以保住客户。

如同 COM 之上的每项技术，OLE 可被概括为一组预定义的 COM 接口。OLE 所需的几项关键技术由 COM 服务提供。这包括结构化存储、绰号、包含拖放的统一数据传输、可连接对象和自动化支持。

OLE 复合文档的方法对文档容器和文档服务器进行区分。文档服务器提供某种内容模型和显示、操作内容的能力。文档容器没有自己的内容，但可以接受任意文档服务器提供的内容成分。许多文档容器也是文档服务器，这即是说，它们支持外来的成分，同时也有自己的内容。大多数流行的“重家伙”，像微软的 Office 应用（如 Word、Excel 和 PowerPoint 等），是结合为一体的服务器和容器。例如，Excel 有自己的内容模型，它是按照电子数据表排列的数据和公式单元。Excel 也是容器，作为容器，它能接受所插入的 Word 文本对象。

10.3.7 .NET 框架

.NET 框架是更大的.NET 空间的一部分。它包含的内容有通用语言运行环境，许多部分接口化和基于类的框架（被打包成配件），以及许多工具。CLR 是通用语言基础设施规范的实现，它增加了 COM+互操作和 Windows 平台访问服务。特别地，CLR 提供了动态载入和卸载、垃圾回收、语境截取、元数据自省、远程化、持久性，以及其他完全和语言无关的运行时刻服务。目前，微软在 CLR 上支持 4 种语言：C#、JScript、Managed C++和 Visual Basic .NET。

配件（Assemblies）是.NET 中部署、版本控制和管理的单元，也就是说，它们是.NET 的软件构件。“并排”使用同一配件的多个版本是完全可以的。配件包含元数据、模块和资源，所有这些以平台无关的方式被表达。模块中的代码以 CIL（通用中介语言）表达，CIL 大致像 Java、Smalltalk 的字节码，或者 Pascal P 码。与早期字节码格式不同，配件

中使用的语言不重视解释。MSIL（微软中介语言）是与 CLI 兼容的超集，它带有支持 CLR 互操作特性的指令，这些特性就在 CLI 规范之外。CLR 在安装或者载入时被编译，执行的始终是本地码。CLR 自省和其他基于类型的概念覆盖了很大的类型系统空间，此空间被称做 CTS（通用类型系统）。

下面涵盖了各种与.NET 框架相关的技术细节。

1．.NET 大图景

微软公司的.NET 计划的目标是，将范围广泛的微软产品和服务组织起来，置于各种互联设备共同的视野范围内，这些设备包括服务器、固定和移动 PC 及特殊设备。在技术层次，.NET 瞄准如下三个层面。

（1）Web 服务。

（2）部署平台（服务器和客户机）。

（3）开发平台。

Web 服务想达到因特网的传递式可编程性（这就不仅仅是传统意义上瞄准人类客户的 Web 了，它应包括支持 Web 服务的构造、定位和使用的因特网和 Web 的标准和建议）。为启动 Web 服务空间，微软公司计划推出许多基础的核心服务。第一个这样的服务已推出一段时间，它就是用于验证用户的.NET 护照。另一个是.NET 警报，它在 2002 年早期被应用。它是通用警报服务，在引入时是通过 Windows 信使发送警报的。作为.NET My Services 和其他计划的一部分，微软公司公布了更多的服务，例如用于存储的服务。从各式服务器产品和 Windows .NET 服务器开始，微软平台正在经过一系列步骤被转变，以便以本地和有效的方式支持、使用 Web 服务和处理 XML。

最后，也是本章的焦点，会有新的开发平台，它包含 CLR、框架和工具。CLR 提供了新的构件基础设施，可以（但不是必须）为构件屏蔽底层硬件平台的细节。类似 JVM，CLR 定义了一套脱离具体处理器的指令集。与 JVM 不同的是，CLR 还支持需要和特定底层平台紧密集成的构件。

2．通用语言基础设施

通用语言基础设施规范由微软公司、英特尔公司和惠普公司联合提交给 ECMA，它建立了类似 CORBA 的语言中性平台。可是与 CORBA 不同，CLI 也定义了中介语言（Intermediate Language，IL）和部署文件格式（配件），例如 Java 字节码、类和 JAR 文件。与 CORBA 和 Java 不同，CLI 支持可扩展元数据。通用语言运行环境是微软.NET 框架的一部分，它是微软公司对 CLI 规范的实现。CLR 超出了 CLI 兼容的范围，它包括对 COM 和平台互操作的支持（细节参见下个小节）。CLI 包括了执行引擎服务的规范（例如载入器、JIT 编译器、起垃圾回收作用的内存管理器）、通用类型系统（Common Type System，CTS）和通用语言规范（Common Language Specification，CLS）。CTS 和 CLS 起着互补的作用，CTS 范畴是许多语言在类型空间的核心概念的超集。与 CLI 兼容的代码能够在整个 CTS 空间运行。可是，没有哪两种语言能精确覆盖相同的 CTS 子集。以

不同语言实现的代码要互操作，CLS 空间就显得有用。CLS 是 CTS 的严格子集，它被构建的方式使许多语言都完全覆盖它。特别地，若某个定义是与 CLS 兼容的，那么任何被归为 CLS 消费者的语言均能使用该定义，这是 CLI 目标语言中最简单而有用的一类。能在 CLS 空间引入新定义的语言称做 CLS 生产者，能扩展 CLS 空间已有定义的语言称做 CLS 扩展者。CLS 扩展者也总是 CLS 生产者，CLS 生产者也总是 CLS 消费者。CTS 为所有类型定义了单根类型——System.Object。Object 之下，CTS 区分了值类型和引用类型。所有值类型是 System.ValueType 的单态子类型，它本身又是 System.Object 子类型。引用类型被分成接口、类、数组和代理（技术上，接口被建模为 CTS 中特殊的类），其中的类被分成按值排置（Marshal-by-value）和按引用排置（Marshal-by-reference）两种。按引用排置又被分成随环境变化和与环境绑定两种。从图 10-10 可看到 CTS 类型体系的概况。

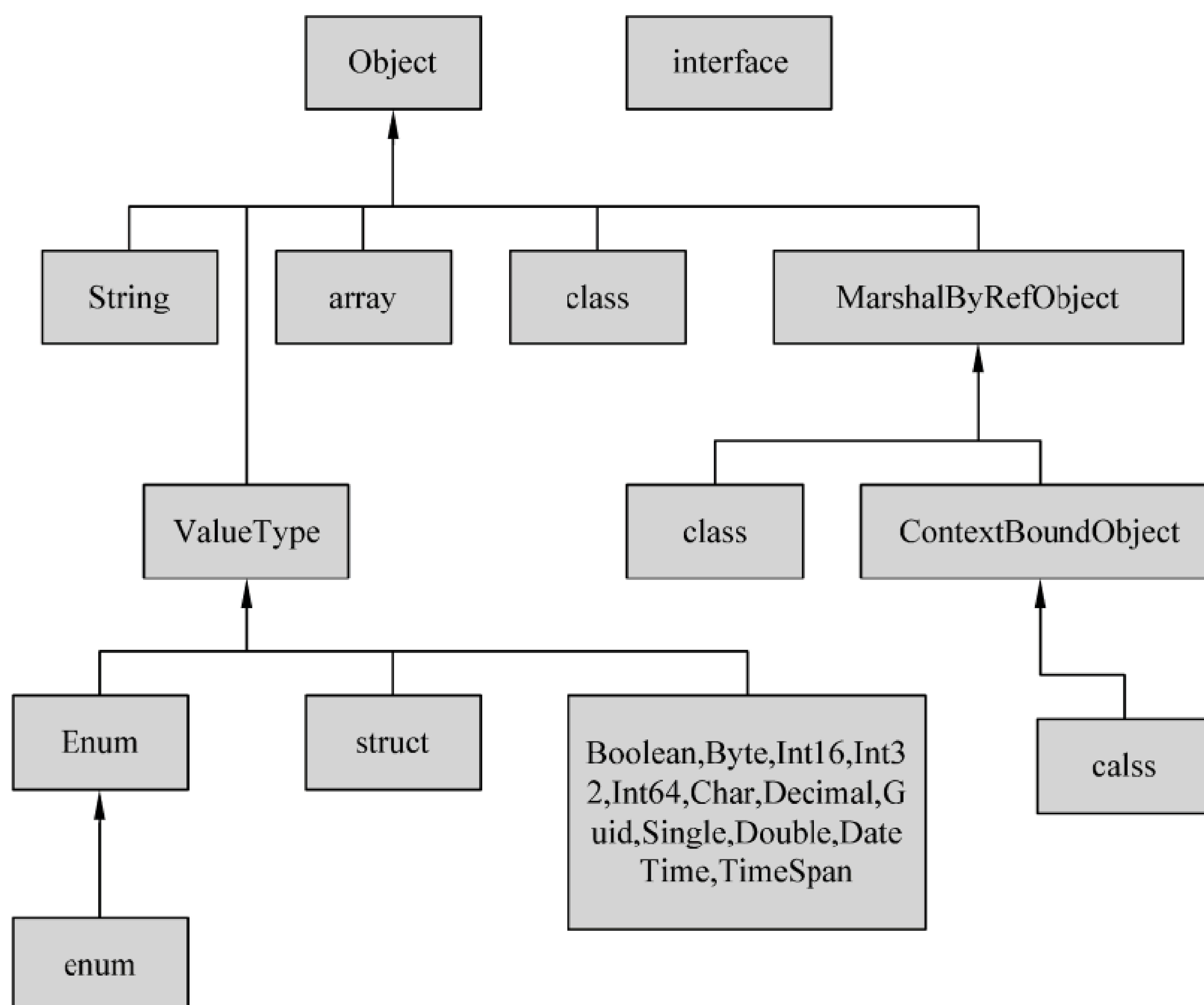


图 10-10 顶层 CTS 类型层次

原始类型是没有的，因此，诸如整型或浮点等类型只是预定义的值类型。多个接口和单个类继承关系是被支持的，甚至值类型也能继承（实现）多个接口。访问权限从两个方面被控制，即定义点和使用点是否位置相同，以及定义点和使用点是否通过类继承而相关。为了前者，区分了三种位置范畴：类、配件和全局。因此，访问权限关系有 6 种可能的约束组合，但大多数语言支持的只是其子集。例如，C#不支持把 protected 访问权限定义在小于全局的范畴上。某些语言，像 Managed C++，则支持所有组合。

方法可以是静态的、与实例绑定的或者虚拟的（虚拟也暗示着是与实例绑定）。对重载的支持要依靠方法名、签名，但没有返回类型。重载的解析策略依语言而定（因此，CLI 自省机制引入了自己的重载解析策略）。

类能实现多个接口，并可以用引入的接口名修饰方法名。因此，能够在相同的类上实现两个接口，即使它们包含具有相同名字和特征的方法，但这两个方法应以不同的方式实现。例如，C#完全支持显式地实现接口方法的概念。

```
interface IShape{
void Draw( );
}
interface ICowboy{
void Draw( );
}
class CowboyShape:IShape,Icowboy{
 void IShape.Draw(){}
voidICowboy.Draw(){}
}
```

尽管上面的 cowboy/shape 例子被广为使用，但实际上它没有包含一个重要情况：偶然的名字冲突会发生这个问题。然而，重要得多的是这样一个例子：发布新的接口版本时还希望能并排支持多个构件。图 10-11 显示了类 C 应该如何实现版本 1 的接口 I 和版本 2 的接口 I，以便和类 A 和 B 正确交互，此二者都需要接口 I，只是版本不同。CTS 把所有的名字定义锚定在包含它们的配件的名字里。因为配件的名字包含了版本信息，接口 I 的两个版本实际上可以被区分开，不过它们的方法名还有可能冲突。CTS 支持在同样的类上实现接口的两个版本，这是支持并排使用配件多个版本的重要步骤。

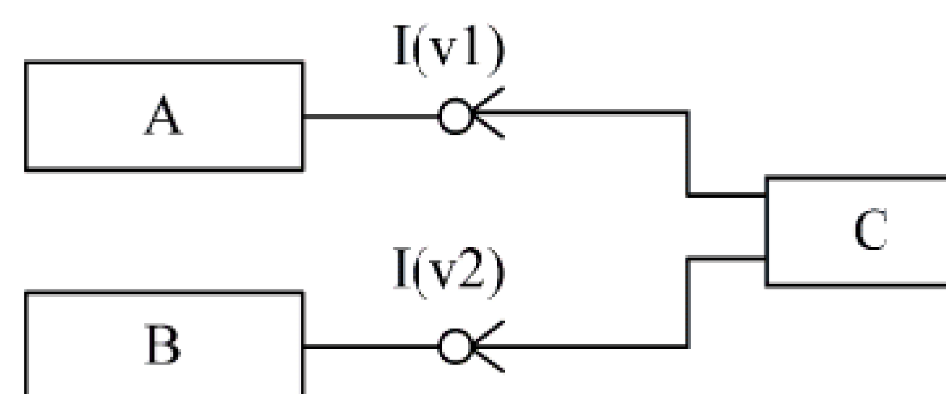


图 10-11　一个接口不同版本的并行实现

各种方法命名规范，例如属性和索引器的取值-设值方法，也是 CTS 的一部分。定义这些规范的目的是支持跨语言互操作，而没有考虑对各种语言中传统命名的特性的显式支持。例如，对于 C#属性 Foo，对应的访问方法被称做 get_Foo 和 set_Foo。C#不允许直接使用这些方法名，但其他不直接支持属性的语言就可以调用这些方法来访问属性。

可抛出异常不是 CTS 方法特征的一部分。与 Java 和 C++不同，CTS 没有规定调用方法时要静态检查可抛出的异常。语言仍可以在它们自己领域内自由执行这类检查。没有哪两种语言对声明和检查的语义能达成完全一致，这使得跨语言检查没什么用处，即便 CTS 曾支持注释。

3．COM 和平台的互操作

CLR 能真正支持与 COM 的互操作，并能直接访问底层平台（即 Win32 和其他基于

DLL 的 API)。通过整合对 COM 互操作和平台调用的支持，CLR 执行引擎能提供几乎最优的性能。例如，平台调用通过 JIT 被编译为本地代码序列，它实际上和传统编译的代码是等同的。COM 互操作是通过提供两类自动合成的包装达到的。一种是 COM 可调用包装，它通过 COM 接口来呈现 CLR 对象；另一种是运行环境可调用包装，它通过 CLR 接口呈现 COM 对象。

为了和 COM 互操作，CLR 工具可被用来创建互操作配件，这些配件定义的类型和 COM 类型库定义的类型相匹配。被多个.NET 应用共享的 CLR 配件必须有唯一的强名字，这对 COM 互操作有微妙的影响。正是这点使多方可能为相同的 COM 接口（相同的 IID）产生互操作配件。然而，得到的互操作配件会公布相互不兼容的类型，尽管所有这些类型对应着有相同 IID 的相同 COM 接口。为避免此种情况，定义了主互操作配件（Primary Interop Assemblies，PIA）。PIA 应由 COM 接口（IID“所有者”）的发布者产生。如果得不到 PIA，可以产生替代的互操作配件，只是它的类型只能在配件内部使用。把这些类型公布在新的配件特征中，最终会导致与其他配件的不兼容，它们或者依赖 PIA，或者将自己的依赖公布于另一个替代的互操作配件。

尽管 COM 表面简单，但是 COM 互操作却是微妙而复杂的。原因是，COM 调用规范的细节（包括分派接口），排置规范（包括由谁分配或释放的规则），以及对底层不安全类型的支持（包括指向大小未知的数组的指针）。远程化接口（DCOM 代理能自动生成的接口）更容易处理，因为代理需要的是和 CLR 包装所需大致相同的信息。不幸的是，DCOM 给自己带来了麻烦——类似[call_as()]属性的 IDL 注释只有过程意义，不能被自动解释，以产生合适的 CLR 包装。

奇怪的是，若接口被限制为“同构类型”（此类型在穿过 CLR/COM 边界时无需变换），调用代码中的 COM 方法时，其开销大约只有 50 个指令周期。

10.4 战略比较

迄今为止，讨论过的每个方法都已给出了丰富的技术细节，那么它们有什么显著的区别和基本的共同点呢？战略上的后果又是什么呢？

10.4.1 共性

很明显，所讨论过方法的共性不能帮助决定应该遵循哪一种方法。然而不管具体的方法是什么，这些方法的共性有助于我们做出使用构件化软件技术的决定。对这些共性的理解还可以避免一些没有意义的争论，这些争论的产生是因为把一些次要问题简单地误解为主要的不同点了。

所有的方法都依赖于延后绑定机制、封装和动态多态性（也被称为包含多态性或者子类型化）。除了 COM 之外，其他方法都支持接口继承（COM 中的多态性来源于接口

及类的分离及每个类的多接口支持）。换一个说法就是，所有的方法都依赖于某种类型的对象模型。

另外，随着时间的推移，这些方法互相取长补短。大多数的方法目前支持：

（1）一种构件传输格式——JavaJAR 文件、COMcab 文件、CCM:-、CLI 配件。

（2）统一方式的数据传输。

（3）事件和事件连接或者信道、单播和多播。

（4）元信息——自省、反射。

（5）某种形式的持久化、序列化或者外部化。

（6）基于属性的编程或部署描述符。

（7）适合于应用服务器的特定构件模型——EJB、COM+、CCM 和 CLR:COM+。

（8）适合于 Web 服务器的特定构件模型——JSP/servlets、COM:-、CCM:-和 ASP.NET。

一个通常被忽视的事实是，非 COM 方法慢慢地汇聚到同样支持 COM 方法已经拥有的功能上来，这个功能就是构件对象通过多个截然不同的对象将自身展现给客户的构件对象。这样做使得动态配置成为可能，这已经得到了认可。CORBA 构件模型的等价接口的概念几乎等同于 COM 的 QueryInterface 方法。JavaBean 引入了间接的库 java.beans.Beans，来代替 Java 语言的类型测试（instanceof）和类型检查（检查转换）。通过这种方法，将来的 bean 可以将它们自己作为一组 Java 对象表示给客户，而不是一个单独的对象。有趣的是，CLI/CLR 在第一个版本中没有追随 COM，不提供对处理拥有多个实现体的实例的支持和转换，尽管提供了通用的设计模式，使用 C#的属性从主对象中获得子对象。

10.4.2　不同点

一旦做出使用软件构件的决定，下一步就要选择使用哪一种方法。鉴于很多方法具有较多的共性，可以同时遵循几种不同的方法。特别地，支持者们和第三方或许会提供主要方法之间的互连的解决方案。这里有若干例子。IONA 公司的 Orbix 2000 COMet 是一个 CORBA/COM 集成工具。Sun 公司的 ActiveX bridge 允许 JavaBean 实例被嵌入到 ActiveX 容器中。IONA 公司的技术总监 Annrai 说：“我们的座右铭是，不兼容就意味着商业机会，对于我们而言是巨大的机遇。”

以下是一个这些方法间明显区别的（不全面的）列表。

1．每个平台的二进制接口标准

构件交互的二进制标准是 COM 的核心（值得注意的是，虽然从技术上是可行的，也曾经做过尝试，但是 COM 从来没有脱离过 Windows 领域。从而，有一个平台就有一个二进制标准）。Java 通过标准化字节码来避免实际的二进制标准。Java 为二进制接口定义了 Java 本地接口（JNI），JNI 的设计是基于 COM 的，不过是特定于 Java。特别的是，其设计为支持现代的垃圾搜集器提供了空间。CORBA 仍然没有定义二进制标准。

二进制标准是 Direct-to-*编译器需要的，这些编译器将一种特定编程语言的语言构造直接映射到二进制接口。CLR（CLI 的超集）和 Java 类似，采用标准化 MSIL（CIL 的超集）来代替二进制标准。CLR 支持了领先时代的编译，提供了对平台 API 调用进行转换的有效支持和与 COM 的互操作。

2．兼容性和可移植性的源代码级的标准

CORBA 在标准化语言绑定方面做得相当好，语言绑定保证了跨 ORB 实现的源代码的兼容性。大量的标准化的服务接口增强了它在这方面的地位。目前在对象服务器上对 ORB 特定的功能进行存取的实践，降低了基于 CORBA 服务器的可移植性。对于 Java，对 Java 语言规范达成一致，即只要没有其他语言在 Java 平台上被使用，就解决了这个问题。于是语言绑定的标准化成为一个问题，不然由多种源语言产生的字节码之间的互操作将受到危害。Java 包含了越来越多的（事实上是 Sun 的）标准，特别是其中的 J2EE 标准受到很多厂商的追随并提供了实现。COM 没有任何源代码级标准或标准语言绑定的概念。COM 接口市场的标准也没有超出微软事实上的标准。.NET CLR 提供了通用语言规范（Common Language Specification，CLS）来指导语言绑定，它在没有规定单个语言绑定的情况下达到了很高程度的互操作性。位于 CLR 之下的公共语言基础设施（Common Language Infrastructure，CLI）规范，以及一组基框架和 C#语言，都已经由 ECMA 进行了标准化。

3．逐渐形成的和仓促造就的标准

在被制定成“标准”之前，COM、CORBA、Java 和 CLI 标准（按照这个次序）经历了越来越短的演化期。COM（具有 OLE 1 的）与 CORBA（1.2）已经经过了不少实质性的修订，已经没有了向后的兼容性。COM/OLE/ActiveX 有许多冗余的机制——例如外出接口和可连接对象（又被称为变化通知接口，也被称为 advice 接口）与分派接口（又叫做 verb 接口，在 ActiveX 出现以后，还被称为 command target 接口）。不同演化期长度的一个结果就是这些方法在市场上产品数量的不同。市场上有成千上万的 ActiveX 对象，而只有很少的 Bean。然而，EJB 构件已经在工业界获得了实质性的支持，虽然目前大多数的 EJB 构件只是在内部被开发和使用。对于 CLI/CLR 标准的构件，报告其市场的接受程度还为时过早。

4．内存管理、生命周期和垃圾回收

目前 CORBA 尚未提供解决分布式对象系统中的全局内存管理问题的一般方法。COM 和 DCOM 完全依赖于引用计数——这在所有构件都遵循特定规则的情况下是可行的，但是在大的开放的分布式系统中存在伸缩问题。Java 完全依赖于垃圾回收机制，JDK 1.1 以后通过使用 Java RMI 也定义了分布式对象模型并支持分布式垃圾回收，这个概念基于“租期”——预先指定远程引用的生命周期。CLR 也采用垃圾回收机制并融入基于“租期”的对远程引用的生命周期控制。另外，CLR 支持其他的通信和列集协议，诸如 HTTP 之上的 SOAP。

5. 容器管理的持久性和关系

EJB 创新性地引入了容器管理的持久性技术，并且从 EJB2.0 开始，还引入了容器管理的关系。CCM 也有类似的技术，因为它可以算是 EJB 的超集。迄今为止，COM+和 CLR 都尚未提供这样的支持。这些机制仍然需要进行改进，例如一个 J2EE 服务器中过度热心地装载一个关系中的所有实体，导致了很多应用程序的性能低下。OLEDB（及 COM+和 CLR）支持可插拔的持久性映射，允许将数据保存到除数据库外的其他多种外部存储的持久性。2.0 版本的 EJB 不包括对可插拔映射的支持，使得纯数据库应用程序外的容器管理的持久性和关系功能很弱。同样地，当映射需要复杂的连结或存储过程时，EJB 2.0 的局限性也很大。

6. 演化和版本的概念

COM 坚持一旦接口和它们的接口 ID 被公布之后，就必须冻结接口和接口的规约。这样可以解决版本和移植问题，但在某些特定的部署场所使用受控的版本兼容策略时会暴露出一些问题。CORBA 没有直接处理这个问题，而是选择支持主、次版本号这样一个较弱的概念。CORBA 的解决方案是有问题的，因为它允许某个版本的对象的引用被传递给另一个希望接收不同版本对象引用的对象——版本检查只在对象创建的时候进行。Java 只在二进制兼容性级别上考虑版本，为此给出了令人头痛的规则列表。有些规则的实际意义可能不大。例如，一个版本中某个常量的值在另一个版本中被改变，这对于以前编译过的客户程序不会有影响，它们只需简单地保持以前的值就可以了。虽然客户程序使用了不同版本，但是原有的客户程序仍然是可用的（尽管可能会出现一些功能失常），而不用声称该客户程序被破坏。构件 Pascal 的实现使用指纹标注每个接口的算法来保持小粒度上的兼容性（Crelier，1994）。CLI 拥有最完整的版本控制方法。CLI 构件，被称为配件，都标记它们自己的版本信息，以及它们依赖的构件集合的所有版本信息。策略可以用来建立匹配版本的可容忍的范围。通过支持并行运行来允许一个构件的多个版本同时存在，这使得滑动窗口方式的移植成为可能——不是每个构件都需要立刻更新到一个新的版本。然而，最初的.NET 框架和面向 CLR 的语言都没有利用 CLI 版本支持的全部优势。

7. 分类的概念

COM 中的分类通常被忽视，因为这个概念比较新，并且看起来没有什么坏处，不过实际上，它引入了合约绑定到包含任意多个接口的规约这样的概念。一个构件可以属于任意数目的分类，一个框架或者其他构件可以使用分类成员资格作为高层的断言。Java 和 CORBA 没有任何类似的概念，虽然 Java 中的空标记接口按照类似的目的被使用。CLI 提供定制属性来扩展构件的元数据，因此分类和其他元信息可以使用定制属性来获取。

8. 产业界的实现支持及应用状况

这里，所有的方法有它们自己的领地。COM 在客户机/桌面系统方面最强。J2EE 和 COM+则在基于非 PC 和基于 PC 的服务器的解决方案中占主导地位。Web 服务器主要使

用 JSP 或者 ASP（现在还有 ASP.NET）。CORBA 在商业计算层次上对传统的遗留系统的集成是最强的。COM 和 CLR 很大程度被限制在微软所提供的实现上。很多厂商提供了 CORBA 和 J2EE 的实现。从一个 J2EE 服务器移植到另一个 J2EE 服务器并不是一件容易的事情，但是当然要比 J2EE 和.NET 间的移植简单很多。

9．开发环境

支持 COM 的开发环境相当多。Java 的开发环境也比较多。支持 CORBA 的开发环境非常少，几乎没有。对于 CLR——微软对 CLI 的实现，其开发环境是与之一起发布的 Visual Studio .NET，该环境包括对 Visual Basic、JScript、C#和 Managed C++的支持。

10．服务

CORBA 目前拥有全套的标准化服务，不过其中的大部分缺少商业实现。COM+用一组丰富的关键服务对 COM 进行了补充，其中包括事务和消息。包括 EJB 的 J2EE 也拥有相对丰富的服务。CLR 用 COM+提供高度的互操作性支持，包括所有的 COM+服务（现在被称为企业服务）。然而，这些服务没有被 CLI 规范所涵盖。在未来，一些 COM+服务可能发展成为真正的基于 CLR 的服务。在 CORBA 和 COM+中提供了对分布式事务协作的支持（对于 CLR 也是同样的），不过并不包括在 EJB 2.0 标准的范围中。J2EE 服务器的支持则会相应的不同。

11．部署

J2EE、COM+、CCM 和 CLR 全都遵循基于属性编程的 MTS 概念。EJB 将属性分离出来并将它们放置在单独的基于 XML 的部署描述符中，使得在特定的部署步骤中可以拥有清晰的操作对象。J2EE 将部署描述符的概念扩大到若干个构件模型。CLR 将基于 XML 的配置和基于 CLI 的定制属性组合到一起。定制属性简化了代码的排列，作为属性的元数据被直接存放在相应的源代码中。这样划分了开发者（放置定制属性）和部署者（处理配置文件）两者的任务。

12．网络服务构件

CORBA 和 COM 在这方面没有特定的构件模型。J2EE 有 JSP 和 servlet 构件。.NET 框架有 ASP.NET 的页面构件类。JSP 在某些方面遵循以前的 ASP 模型，但是在 JSP 页面与 servlet 方面有所改进。ASP.NET 在某些方面遵循 JSP 模型，不过使用目标独立的方式，从而作为取代提供生成 HTML 的构件的方式，ASP.NET 鼓励使用已有的生成界面显示的构件。因此，很多 ASP.NET 构件连一行 HTML 也不直接生成，使得它们独立于特定目标设备的要求，例如为移动设备生成 WML 显示。

13．传输

CORBA 支持 IIOP，用来作为 ORB 之间互操作的标准的传输协议。另外，OMG 采用 XML 和 XML Schema 规范作为应用程序层的传输格式描述。Java 支持 IIOP 绑定，不过也自然支持它自己的 RMI 协议。Java 对 XML 的支持正在改进中。COM 使用 DCOM 作为自身的传输协议，COM+增加了对多种消息格式的支持。CLR 延续了 COM 和 COM+所支持的所有格式，并添加了对 XML schema 定义和 SOAP 调用协议的支持。

第 11 章　信息安全技术

11.1　信息安全关键技术

11.1.1　加密和解密技术

计算机网络的广泛应用，产生了大量的电子数据，这些电子数据需要传输到网络的许多地方，并存储起来。对于这些数据，有意的计算机犯罪和无意的数据破坏成为最大的威胁。原则上来说，对电子数据的攻击有两种形式：一种称为被动攻击，就是非法地从传输信道上截取信息，或从存储载体上偷窃、复制信息。另一种称为主动进攻，就是对传输或存储的数据进行恶意的删除、篡改等。实践证明，密码技术是防止数据攻击的一种有效而经济的方法。

我们把消息的发送者称为信源；消息的目的地称为信宿；没有加密的消息称为明文；加密后的消息称为密文；用来传输消息的通道称为信道。通信时，明文 M 通过变换 E 得到密文 C，即 C=E（u，v，…，w；m）。这个过程称为加密，参数 u，v，…，w 称为密钥。这里所说的变换 E，就是加密算法。从密文 C 恢复到明文 M，这个过程称为解密。解密算法 D 是加密算法 E 的逆运算。

加密密钥与解密密钥相同，或者加密密钥与解密密钥可以简单相互推导的密码体制称为对称密码体制。现代密码学修正了密钥的对称性，加密、解密密钥是不同的，也是不能（在有效的时间内）相互推导的，称为非对称密码体制。

1．对称密钥密码体制及典型算法

对称算法（Symmetric Algorithm），有时又称为传统密码算法，在大多数对称算法中，加密密钥和解密密钥是相同的，所以也称秘密密钥算法或单密钥算法。它要求发送方和接收方在安全通信之前，商定一个密钥。对称算法的安全性依赖于密钥，泄漏密钥就意味着任何人都可以对他们发送或接收的消息解密，所以密钥的保密性对通信至关重要。

对称加密的优点在于算法实现的效率高、速度快。对称加密的缺点在于密钥的管理过于复杂。常用的对称加密算法有 DES、IDEA 等。

1）DES 算法简介

DES（Data Encryption Standard，数据加密标准）是由 IBM 公司研制的一种加密算法，美国国家标准局于 1977 年公布把它作为非机要部门使用的数据加密标准。二十年来，它一直活跃在国际保密通信的舞台上，扮演了十分重要的角色。

DES 是一个分组加密算法，它以 64 位为分组对数据加密；同时 DES 也是一个对称

算法。它的密匙长度是 56 位（因为每个第 8 位都用作奇偶校验），密匙可以是任意 56 位的数，而且可以任意时候改变。其保密性依赖于密钥。

DES 算法分如下 3 个步骤进行。

（1）对给定的 64 位的明文 x，通过一个初始置换函数 IP 来排列 x，从而构造出长为 64 位的串 X_0，记 X_0 为 IP（X）$=L_0R_0$，L_0 表示 X_0 的前 32 位，R_0 表示 X_0 的后 32 位。

（2）计算 16 次迭代，设前 $i-1$ 次迭代结果为 $X_{i-1}=L_{i-1}R_{i-1}$，则第 i 轮迭代运算为：

$$L_i=R_{i-1} \qquad R_i=L_{i-1}\oplus f\,(R_{i-1},\ k_i)$$

其中，L_{i-1} 表示 X_{i-1} 的前 32 位，R_{i-1} 表示 X_{i-1} 的后 32 位，$\oplus$ 表示两位串的“异或”运算，f 主要是由一个称为 S 盒的置换构成。K_i 是一些由初始的 56 位经过密钥编排函数产生的 48 位长的块。

（3）对位串 $L_{16}R_{16}$ 作逆置换 IP^{-1} 得密文 y，$y=IP^{-1}$（$R_{16}L_{16}$），置换 IP^{-1} 是 IP 的逆置换。

DES 算法的示意图如图 11-1 所示。

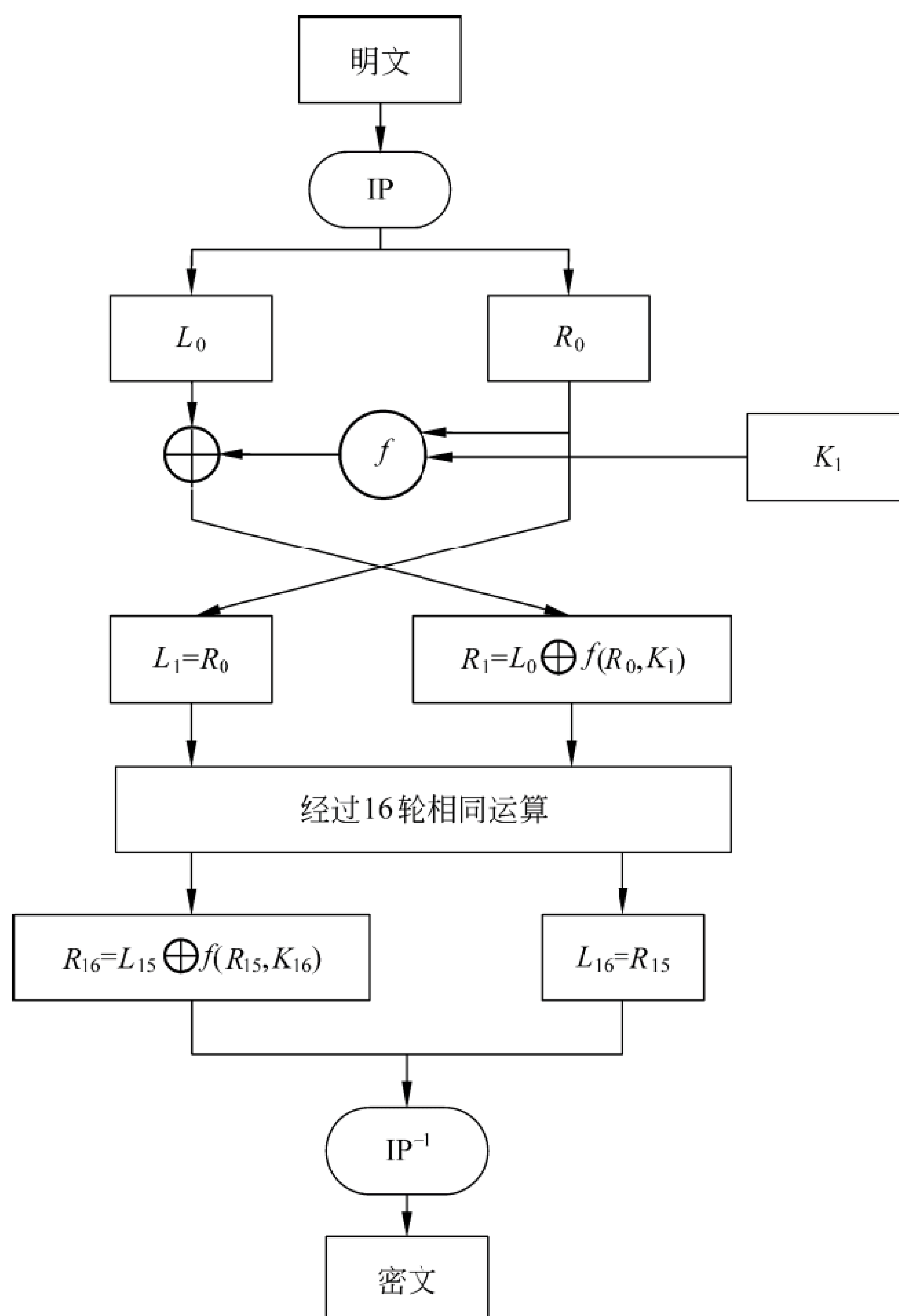


图 11-1　DES 算法示意图

2）IDEA 算法简介

国际数据加密算法（International Data Encryption Algorithm，IDEA）是 1992 年来学嘉等人设计的算法。该算法的前身是 1990 年公布的推荐加密标准（Proposed Encryption Standard，PES）算法。

IDEA 分组长度为 64b，密钥长度为 128b。其使用的运算非常简单，只需要异或，模 2^{16} 加和模（$2^{16}+1$）乘，这些算法都很容易使用硬件或者软件实现（而 DES 算法便于用硬件实现，难以用软件实现），所有运算都是使用基于 16b 数运行，很容易在现在的 16b、32b、64b CPU 上实现。由于这一特性，使用软件实现的 IDEA 算法的运算速度比 DES 算法要快。由于 IDEA 算法使用的密钥长度为 128b，远远大于 DES 算法的 56b，对于 128b 密钥来说，使用穷举法攻击的方法是不现实的。

2．不对称密码加密算法

不对称密码体制又称为双密钥和公钥密码体制，是于 1976 年由 Diffie 和 Hellman 提出的。与对称密码体制相比，非对称密码体制有两个不同的密钥，其中一个密钥称为私钥，该密钥被秘密保存；另一个密钥公开，不需要保密。

公钥密码系统的工作方式为：任何人都可以将自己加密的公钥公布在网络或其他可以公开的地方。其他人欲传送信息给该接收方时，可使用该接收方所公布的公钥将信息加密之后传送给接收方。接收方收到加密后的信息时，就可以利用拥有的与此公钥相对应的私钥，将该加密信息解出来。所以公开密钥密码系统的通信双方，不需要事先通过安全秘密管道交换密钥，即可进行通信。

RSA 密码体制是一个常用的非对称的密码体制，它是一个既能用于数据加密也能用于数字签名的算法。

RSA 的安全性依赖于大素数分解。公钥和私钥都是两个大素数（大于 100 个十进制位）的函数。据猜测，从一个密钥和密文推断出明文的难度等同于分解两个大素数的积。

1）密钥对的产生

（1）选择两个大素数，p 和 q。

（2）计算 $n=p*q$。

（3）随机选择加密密钥 e，e 必须满足以下条件：

$$\text{GCD}(e,\ \Phi(N))=1$$

其中，Φ为 Euler’s Function，$\Phi(N)$ 为小于 N、且与 N 互质的整数的个数。在此，$\Phi(N)=(p-1)*(q-1)$（也有些做法是取 $\text{LCM}((p-1)*(q-1))$）。

（4）利用 Euclid 算法计算解密密钥 d，满足 $d=e^{-1} \bmod \Phi(N)$。

产生出加密公钥 e、N 与解密密钥 d 之后，使用者将 e 及 N 公开，就可以使用它们来执行加解密的工作了。

2）加密程序

使用者将其欲加密的信息 M，在取得对方的公钥 e 及 N 之后，执行模（mod，即同

余的运算，$C=M \bmod N$，C 等于 M 除以 N 的余数）指数运算，就可获得密文 C。

$$C=M^e \bmod N \quad \text{（a）}$$

然后通过网络传送至通信的对方。

3）解密程序

对方在收到密文 C 后，以自己的私钥执行下面的解密程序，解密时作如下计算：

$$M=C^d \bmod N \quad \text{（b）}$$

即可获得明文 M。

RSA 可用于数字签名，方案是用 （b）签名， 用（a）验证。具体操作时考虑到安全性和 M 信息量较大等因素，一般是先作 HASH 运算。

RSA 的安全性依赖于大数分解，由于进行的都是大数计算，使得 RSA 最快的情况也是 DES 百分之一。无论是软件还是硬件实现，速度慢一直是 RSA 的缺陷，因此一般来说，RSA 只用于少量数据加密。

11.1.2 散列函数与数字签名

1．MD5 散列算法

散列函数是一种公开的数学函数。散列函数运算的输入信息叫做报文，运算后所得到的结果叫做散列码或者叫做消息摘要。散列函数 $h=H(M)$ 具有如下一些特点。

（1）不同内容的报文具有不同的散列码，给定 M，要找到另一消息 M，使 $H(M)=H(M')$ 很难。

（2）散列函数是单向的，给出 M，容易计算出 h。给定 h，根据 $h=H(M)$ 反推 M 却很难。

（3）对于任何一个报文，无法预知它的散列码。

（4）散列码具有固定的长度，不管原始报文的长度如何，通过散列函数运算后的散列码都具有一样的长度。

由于散列函数具有这些特征，因此散列函数可以用来检测报文的可靠性。接收者对收到的报文用与发送者相同的散列函数进行运算，如果得到与发送者相同的散列码，则可以认为报文没有被篡改，否则报文就是不可信的。

常见的散列函数有 MD5、SHA 和 HMAC 等。

MD5（Message Digest 5）是一种非常著名的散列算法，已经成为国际标准。它是在 MD4 的基础上改进的算法，是具有更好的安全性能的散列算法。MD5 散列算法对输入的任意长度消息产生 128 位（16 字节）长度的散列值（或称消息摘要）。MD5 算法包括以下 4 个步骤。

（1）附加填充位。首先对输入的报文进行填位补充，使填充后的数据长度模 512 后余 448。如果数据长度正好模 512 余 448，则需增加 512 个填充位，也就是说填充的个数为 1～512 位。填充位第一个位为 1，其余全部为 0。

（2）补足长度。将数据长度表示为二进制，如果长度超过 64 位，则截取其低 64 位；如果长度不足 64 位，则在其高位补 0。将这个 64 位的报文长度补在经过填充的报文后面，使得最后的数据为 512 位的整数倍。

（3）初始化 MD 缓存器。MD5 运算要用到一个 128 位的 MD5 缓存器，用来保存中间变量和最终结果。该缓存器又可看成是 4 个 32 位的寄存器 A、B、C、D，初始化为：

A: 01 23 45 67　B: 89 ab cd ef　C: fe dc ba 98　D: 76 54 32 10

（4）处理数据段。首先定义 4 个非线性函数 *F*、*G*、*H*、*I*，对输入的报文运算以 512 位数据段为单位进行处理。对每一个数据段都要进行 4 轮的逻辑处理，在 4 轮中分别使用 4 个不同的函数 *F*、*G*、*H*、*I*。每一轮以 ABCD 和当前的 512 位的块为输入，处理后送入 ABCD（128 位）。

2．数字签名与数字水印

1）数字签名

数字签名可以解决否认、伪造、篡改及冒充等问题。具体要求：发送者事后不能否认发送的报文签名、接收者能够核实发送者发送的报文签名、接收者不能伪造发送者的报文签名、接收者不能对发送者的报文进行部分篡改、网络中的某一用户不能冒充另一用户作为发送者或接收者。凡是需要对用户的身份进行判断的情况都可以使用数字签名，例如加密信件、商务信函、订货购买系统、远程金融交易和自动模式处理等。

数字签名方案一般包括三个过程：系统的初始化过程、签名产生过程和签名验证过程。在签名产生的过程中，用户利用给定的算法对消息产生签名；在签名验证过程中，验证者利用公开验证方法对给定消息的签名进行验证，得出签名的有效性。

实现数字签名的方法有很多，目前采用得比较多的是非对称加密技术和对称加密技术。虽然这两种技术的实施步骤不尽相同，但大体的工作程序是一样的。用户首先可以下载或者购买数字签名软件，然后安装在个人计算机上。在产生密钥对后，软件自动向外界传送公开密钥。由于公共密钥的存储需要，所以需要建立一个鉴定中心（Certification Authority，CA）完成个人信息及其密钥的确定工作。用户在获取公开密钥时，首先向鉴定中心请求数字确认，鉴定中心确认用户身份后，发出数字确认，同时鉴定中心向数据库发送确认信息。然后用户使用私有密钥对所传信息签名，保证信息的完整性、真实性，也使发送方无法否认信息的发送，之后发向接收方；接收方接收到信息后，使用公开密钥确认数字签名，在使用这种技术时，签名者必须注意保护好私有密钥，因为它是公开密钥体系安全的重要基础。如果密钥丢失，应该立即报告鉴定中心取消认证，将其列入确认取消列表之中。其次，鉴定中心必须能够迅速确认用户的身份及其密钥的关系。一旦接收到用户请求，鉴定中心要立即认证信息的安全性并返回信息。

目前已经有大量的数字签名算法，如 RSA、ElGamal、Fiat–Shamir、美国的数字签名标准/算法（DSS/DSA）、椭圆曲线等多种。

2）数字水印

随着数字技术和因特网的发展，各种形式的多媒体数字作品（如图像、视频和音频等）纷纷以网络形式发表，其版权保护成为一个迫切需要解决的问题。数字水印（Digital Watermarking）是实现版权保护的有效办法，如今已成为多媒体信息安全研究领域的一个热点，也是信息隐藏技术研究领域的重要分支。该技术是通过在原始数据中嵌入秘密信息——水印（Watermark）来证实该数据的所有权。这种被嵌入的水印可以是一段文字、标识或序列号等，而且这种水印通常是不可见或不可察的，它与原始数据（如图像、音频和视频数据）紧密结合并隐藏其中，在经过一些不破坏源数据使用价值或商用价值的操作后仍能保存下来。数字水印技术必须具有较强的鲁棒性、安全性和透明性。

（1）典型数字水印系统模型。

图 11-2 为水印信号嵌入模型，其功能是将水印信号加入原始数据中；图 11-3 为水印信号检测模型，用来判断某一数据中是否含有指定的水印信号。

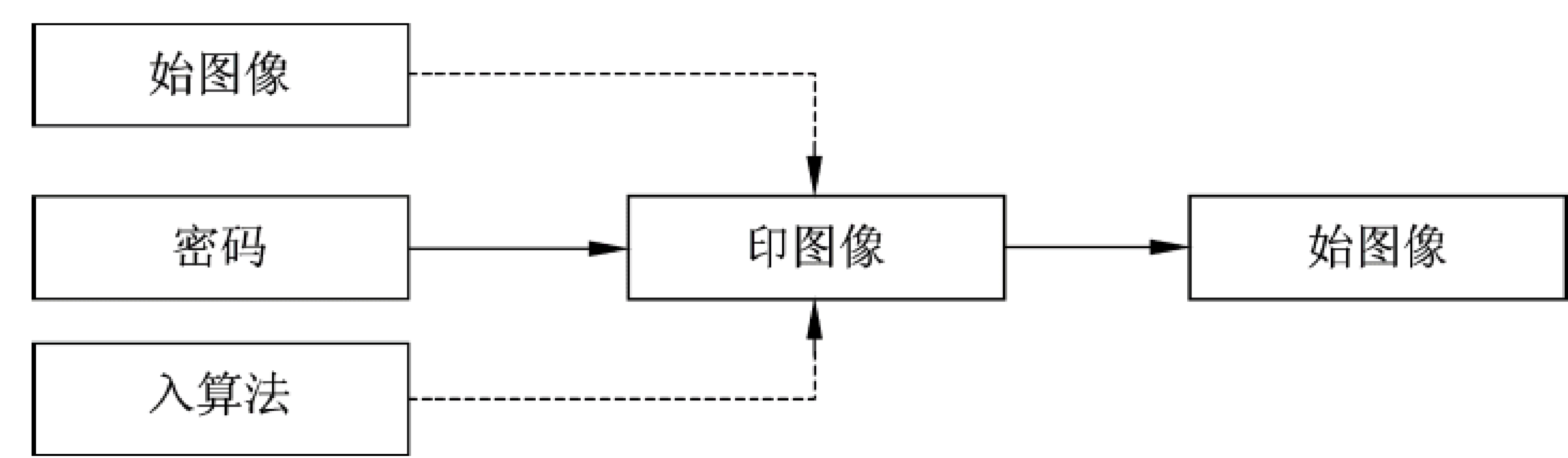


图 11-2 水印信号嵌入模型

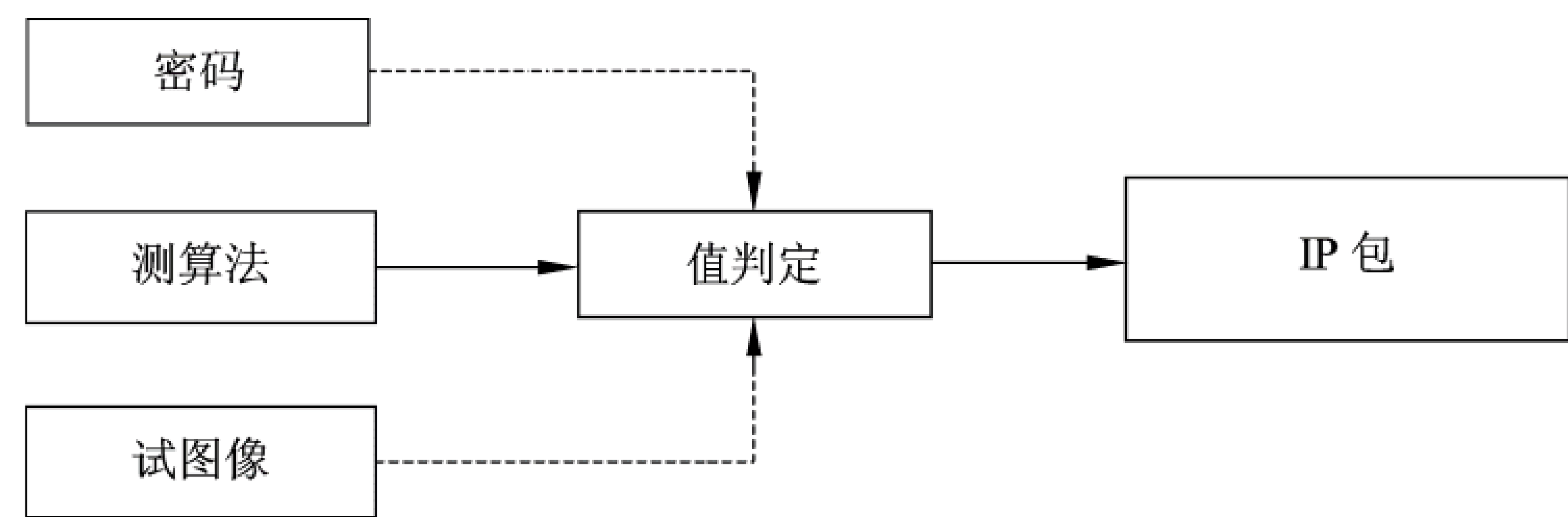


图 11-3 水印信号检测模型

（2）数字水印主要应用领域。

① 版权保护。即数字作品的所有者可用密钥产生一个水印，并将其嵌入原始数据，然后公开发布他的水印版本作品。当该作品被盗版或出现版权纠纷时，所有者即可利用图 11-2 或图 11-3 的方法从盗版作品或水印版作品中获取水印信号作为依据，从而保护所有者的权益。

② 加指纹。为避免未经授权的复制制作和发行，出品人可以将不同用户的 ID 或序列号作为不同的水印（指纹）嵌入作品的合法备份中。一旦发现未经授权的备份，就可以根据此备份所恢复出的指纹来确定它的来源。

③ 标题与注释。即将作品的标题、注释等内容（如一张照片的拍摄时间和地点等）以水印形式嵌入该作品中，这种隐式注释不需要额外的带宽，且不易丢失。

④ 篡改提示。当数字作品被用于法庭、医学、新闻及商业时，常需确定它们的内容是否被修改、伪造或特殊处理过。为实现该目的，通常可将原始图像分成多个独立块，再将每个块加入不同的水印。同时可通过检测每个数据块中的水印信号，来确定作品的完整性。与其他水印不同的是，这类水印必须是脆弱的，并且检测水印信号时，不需要原始数据。

⑤ 使用控制。这种应用的一个典型例子是 DVD 防复制系统，即将水印信息加入 DVD 数据中，这样 DVD 播放机即可通过检测 DVD 数据中的水印信息而判断其合法性和可复制性。从而保护制造商的商业利益。

典型数字水印算法有空域算法、变换域算法、压缩域算法、NEC 算法和生理模型算法等。

11.1.3　密钥分配中心与公钥基础设施

在现代密码系统中，算法本身的保密已经不重要了，对于数据的保密在很大程度上、甚至完全依赖于对密钥的保密。只要密钥能够保密，即使加密算法公开，甚至加密设备丢失，也不会对加密系统的坚固性和正常使用产生多大影响。相反，如果密钥丢失，则不但非法用户可以窃取机密数据，而且合法用户面对密文却如读天书，无法提取有效的信息。因此，在密码系统中，如何高效地分配密钥、安全地管理密钥对保证数据安全来说至关重要。

1．密钥分配中心

一个信息系统中任意两个用户之间都可以自己协商来选择不同的密钥， 显然，对于总共有 N 个用户的系统，每个用户都要保存 $N\times(N-1)$ 个密钥。在用户数量较少时，这样来分配密钥还是比较简单、易用的，但是一旦用户数量多起来，系统中要保存的密钥会急剧增多，让每个用户自己高效、安全地管理数量庞大的密钥实际上是不可能的。

有一种非常有效的密钥自动分配方案是密钥分配中心（Key Distribution Center，KDC）技术。

在 KDC 方案中，每一个用户都只保存自己的私钥 SK 和 KDC 的公钥 PK_{KDC}，而在通信时再从 KDC 获得其他用户的公钥或者仅仅在某一次通信中可以使用的对称密钥加密算法的临时密钥 K。

假设 A 和 B 都是 KDC 的注册用户，他们分别拥有私钥 SK_A、SK_B。设用对称密钥来加密他们之间的这次对话，那么密钥的分配过程如下。

首先，A 向密钥分配中心发送 SK_A（A，B），表示自己想与 B 会话。该请求用自己的私钥 SK_A 加密，KDC 收到 A 的请求，用 A 的公钥来验证请求是由 A 发出的后，根据某种算法来生成供 A、B 之间会话使用的对称密钥 K。KDC 向 A 返回 PK_A（K，PK_B（A，

K)），该应答是用 A 的公钥加密的，只有 A 能解读。A 用自己的私钥解密应答，得到密钥 K，并将 PK_B（A，K）发送给 B，表明 A 欲与 B 进行会话。B 用自己的私钥解密得到会话密钥 K。

至此，完成一次密钥分配。

2．数字证书和公开密钥基础设施

数字签名和公钥加密都是基于不对称加密技术的，存在的问题有：如何保证公开密钥的持有者是真实的；大规模信息系统环境下公开密钥如何产生、分发和管理。

要解决以上问题，就要用到数字证书和 PKI。

1）数字证书

数字证书提供了一个在公钥和拥有相应私钥的实体之间建立关系的机制。目前最常用的数字证书格式是由国际标准 ITU-T X.509 V3 版本定义的。

数字证书中采用公钥体制，即利用一对互相匹配的密钥进行加密、解密。每个用户自己保存私钥，用它进行解密和签名；同时设定一个公钥，并由本人公开，为一组用户所共享，用于加密和验证签名。

数字证书是用户在系统中作为确认身份的证据。在通信的各个环节中，参与通信的各方通过验证对方数字证书，从而确认对方身份的真实性和有效性，从而解决相互间的信任问题。

数字证书的内容一般包括：唯一标识证书所有者的名称、唯一标识证书签发者的名称、证书所有者的公开密钥、证书签发者的数字签名、证书的有效期及证书的序列号等。

2）公钥基础设施

PKI（Public Key Infrastructure，公钥基础设施）的目标是向广大的信息系统用户和应用程序提供公开密钥的管理服务。

PKI 的结构模型中有三类实体：管理实体、端实体和证书库。管理实体是 PKI 的核心，是服务的提供者；端实体是 PKI 的用户，是服务的使用者；证书库是一个分布式的数据库，用于证书和 CRL 的存放和检索。

CA 和 RA 是两种管理实体。CA 是框架中唯一能够发布和撤销证书的实体，维护证书的生命周期；RA 负责处理用户请求，在验证了请求的有效性后，代替用户向 CA 提交。RA 可以单独实现，也可以合并在 CA 中实现。作为管理实体，CA 和 RA 以证书方式向端实体提供公开密钥的分发服务。

持有者和验证者是两种端实体。持有者是证书的拥有者，是证书所声明的事实上的主体。持有者向管理实体申请并获得证书，也可以在需要时请求撤销或更新证书。持有者使用证书声明自己的身份，从而获得相应的权力。验证者确认持有者所提供的证书的有效性和对方是否为该证书的真正拥有者，只有在成功鉴别之后才可与对方进行更进一步的交互。

由于证书库的存取对象为证书和 CRL，其完整性由数字签名来保证，因此不需要额

外的安全机制。

不同的实体间通过 PKI 操作完成证书的请求、确认、发布、撤销、更新和获取等过程。PKI 操作分为存取操作和管理操作两类。其中，存取操作包括管理实体或端实体把证书和 CRL 存放到证书库、从证书库中读取证书和 CRL；管理操作则是管理实体与端实体之间或管理实体与管理实体之间的交互，是为了完成证书的各项管理任务和建立证书链。

11.1.4　访问控制

访问控制是通过某种途径限制和允许对资源的访问能力以及范围的一种方法。它是针对越权使用系统资源的保护措施，通过限制对文件等资源的访问，防止非法用户的侵入或者合法用户的不当操作造成的破坏，从而保证信息系统资源的合法使用。

访问控制技术可以通过对计算机系统的控制，自动、有效地防止对系统资源进行非法访问或者不当的使用，检测出一部分安全侵害，同时可以支持应用和数据的安全需求。

访问控制技术并不能取代身份认证，它是建立在身份认证的基础之上的。

1．身份认证技术

在网络通信中，需要确定通信双方的身份，这就需要身份认证技术。在有安全需求的应用系统中，识别用户的身份是系统的基本要求，认证是安全系统中不可缺少的一部分。识别用户的身份有两种不同的形式：一种是身份认证，要求对用户所有的权限角色或自身的身份进行认证；一种是身份鉴定，要求对使用者本身的身份进行检查。

认证的方法多种多样，其安全强度也不相同。具体方法可归结为 3 大类：根据用户知道什么、拥有什么、是什么来进行认证。用户知道什么，一般就是口令；用户拥有什么，通常为私钥或令牌：用户是什么，这是一种基于生物识别技术的认证。

1）用户名和口令认证

简单认证方式主要是通过一个客户与服务器共知的口令（或与口令相关的数据，如散列、密文等）进行验证。根据处理形式的不同，有 3 种简单认证的方式：验证数据的明文传送、利用单向散列函数处理验证数据、利用单向散列函数和随机数处理验证数据，这 3 种方式的安全强度依次增加，处理复杂度也依次增高。

2）使用令牌认证

在使用令牌进行认证的系统中，进行验证的密钥存储于令牌中。对密钥的访问用口令进行控制。令牌是一个像 IC 卡一样可以加密存储并运行相应加密算法的设备，这种简单认证可以快速、方便地实现用户身份认证，但是认证的安全强度不高。通过令牌可以完成对用户必须拥有某物的验证。令牌的实现分为质询响应令牌和时间戳令牌，其中使用较多的是时间戳令牌。

质询响应令牌的工作原理是：在进行身份认证时，认证服务器首先发送一个随机数到客户机的登录程序。用户将这个随机数读出，输入令牌，并输入令牌的 PIN 码（实际

就是口令），得以访问令牌。令牌对输入的随机数用存储的私钥进行签名，并把结果用Base64编码输出。用户把令牌的输出填入客户机的验证程序中，数据传输到认证的服务器端，在服务器端将使用用户的公钥对签名进行验证，以确定是否允许客户通过登录认证。在该方案中，由于使用数字签名进行登录认证，系统的安全强度大大增加：私钥采用令牌存储的方式解决了私钥自身的安全问题。令牌是一个可移动的设备，可以随身携带，而且令牌有PIN码保护，对令牌的非法访问超过一定的次数后，令牌会死锁。

时间戳令牌解决了质询响应令牌中随机数的问题，时间戳令牌利用时间代替上面的随机数。时间戳令牌每时每刻都在工作，一般每分钟产生一个登录数据，用户只需输入PIN码。登录数据被传送到认证的服务器端，服务器利用当前时间对登录数据进行验证，完成用户的登录过程。使用时间戳令牌需要重点考虑时间同步问题，由于令牌的时钟和认证服务器的时钟不同步，产生的验证码并不会通过验证。解决方法是在验证服务器上进行多次试探验证，在一个时间范围内试探，如果成功则在服务器上存储令牌时钟与服务器时钟的偏移量，以便下次登录时使用。目前，在安全性要求较高的认证系统中，多是采用这种方案。

采用PIN码与令牌实现了双因素验证，根据用户知道什么、拥有什么进行认证，也提供了一个保密认证密钥的方法。但是实现双因素验证需要用户输入数据，给用户的操作增加了麻烦。

3）生物识别与三因素认证

现在兴起了一种基于生物识别技术的认证，主要是根据认证者的图像、指纹、气味和声音等作为认证数据。基于用户知道什么（口令）、拥有什么（私钥和令牌）、是什么（生物特征）的3因素认证是目前强认证中使用最多的手段。在安全性要求较高的系统中，认证必须能对用户进行身份鉴定。要将用户知道什么、拥有什么、是什么结合起来，同时对认证用的密钥进行保护。

2．访问控制技术

根据控制手段和具体目的的不同，通常将访问控制技术划分为如下几个方面：入网访问控制、网络权限控制、目录级安全控制、属性安全控制以及网络服务器的安全控制等。

入网访问控制为网络访问提供了第一层访问控制。它控制哪些用户能够登录到服务器并获取网络资源，控制准许用户入网的时间和准许入网的工作站等。基于用户名和口令的用户的入网访问控制可分为三个步骤：用户名的识别与验证、用户口令的识别与验证、用户账号的默认限制检查。三个步骤中只要任何一个未通过校验，该用户便不能进入该网络。可以说，对网络用户的用户名和口令进行验证是防止非法访问的第一道防线。但由于用户名口令验证方式容易被攻破，目前很多网络都开始采用基于数字证书的验证方式。

网络权限控制是针对网络非法操作所提出的一种安全保护措施。能够访问网络的合

法用户被划分为不同的用户组，用户和用户组被赋予一定的权限。访问控制机制明确了用户和用户组可以访问哪些目录、子目录、文件和其他资源；以及指定用户对这些文件、目录、设备能够执行哪些操作。它有两种实现方式，“受托者指派”和“继承权限屏蔽”。“受托者指派”控制用户和用户组如何使用网络服务器的目录、文件和设备；“继承权限屏蔽”相当于一个过滤器，可以限制子目录从父目录那里继承哪些权限。可以根据访问权限将用户分为以下几类：特殊用户（即系统管理员）；一般用户，系统管理员根据他们的实际需要为他们分配操作权限；审计用户，负责网络的安全控制与资源使用情况的审计。用户对网络资源的访问权限可以用访问控制表来描述。

目录级安全控制是针对用户设置的访问控制，控制用户对目录、文件、设备的访问。用户在目录一级指定的权限对所有文件和子目录有效，用户还可以进一步指定对目录下的子目录和文件的权限。对目录和文件的访问权限一般有 8 种：系统管理员权限、读权限、写权限、创建权限、删除权限、修改权限、文件查找权限和访问控制权限。8 种访问权限的有效组合可以让用户有效地完成工作，同时又能有效地控制用户对服务器资源的访问，从而加强了网络和服务器的安全性。

属性安全控制在权限安全控制的基础上提供更进一步的安全性。当用户访问文件、目录和网络设备时，网络系统管理员应该给出文件、目录的访问属性，网络上的资源都应预先标出安全属性，用户对网络资源的访问权限对应一张访问控制表，用以表明用户对网络资源的访问能力。属性设置可以覆盖已经指定的任何受托者指派和有效权限。属性能够控制以下几个方面的权限：向某个文件写数据、复制文件、删除目录或文件、查看目录和文件、执行文件、隐含文件、共享、系统属性等，避免发生非法访问的现象。

因为网络允许用户在服务器控制台上执行一系列操作，所以用户使用控制台就可以执行装载和卸载模块、安装和删除软件等操作，这就需要网络服务器有安全控制。网络服务器的安全控制包括可以设置口令锁定服务器控制台，从而防止非法用户修改、删除重要信息或破坏数据。具体包括设定服务器登录时间限制、非法访问者检测和关闭的时间间隔等。

11.1.5 安全协议

1．IPSec 协议简述

为了满足 Internet 的安全需求，因特网工程任务组（IETF）于 1998 年 11 月颁布 IP 层安全标准 IP_SECURITY 协议（IPSec），IPSec 在 IP 层上对数据包进行高强度的安全处理提供数据源验证、无连接数据完整性、数据机密性、抗重播和有限通信流机密性等安全服务。

1）IPSec 协议工作原理

IPSec 通过使用两种通信安全协议来为数据报提供高质量的安全性：认证头（AH）协议和封装安全载荷（ESP）协议，以及像 Internet 密钥交换（Internet Key Exchange，IKE）

协议这样的密钥管理过程和协议。其中 AH 协议提供数据源认证、无连接的完整性以及一个可选的抗重放服务。ESP 协议提供数据保密性、有限的数据流保密性、数据源验证、无连接的完整性以及抗重放服务。IPSec 允许系统或网络用户控制安全服务提供的粒度。IPSec 的安全服务是由通信双方建立的安全关联（Security Association，SA）来提供的，SA 为通信提供了安全协议、模式、算法和应用于单向 IP 流的密钥等安全信息。每一个 IPSec 节点包含一个局部的安全策略库（Security Policy Database，SPD），系统在处理输入、输出 IP 流时必须参考该策略库，并根据从 SPD 中提取的策略对 IP 流进行不同的处理：拒绝、绕过、进行 IPSec 保护。如果策略决定 IP 流需要经过 IPSec 处理，则根据 SPD 与 SAD 的对应关系，找到相应的 SA，并对 IP 包进行指定的 IPSec 处理。SA 由一个三元组唯一地标识，该三元组包含一个安全参数索引（Security Parameter Index，SPI），一个用于输出处理 SA 的目的 IP 地址或者一个用于输入处理 SA 的源 IP 地址以及一个特定的协议（例如 AH 或者 ESP）。SPI 是为了唯一标识 SA 而生成的一个 32 位整数。它在 AH 和 ESP 头中传输，IPSec 数据报的接收方易于识别 SPI 并利用它连同源或者目的 IP 地址和协议来搜索 SAD，以确定与该数据报相关联的 SA 或者 SA 束。SA 中所选用的安全协议、SA 模式、SA 的两端及安全协议内所要求的服务等具体地决定了怎样为通信流提供安全服务。但是，最终安全服务的具体实施是通过使用 AH 和 ESP 协议。

2）IPSec 协议实现模式

IPSec 协议既可用来保护一个完整的 IP 载荷，也可用来保护某个 IP 载荷的上层协议。这两方面的保护分别由 IPSec 的两种不同“模式”来提供，如图 11-4 所示。

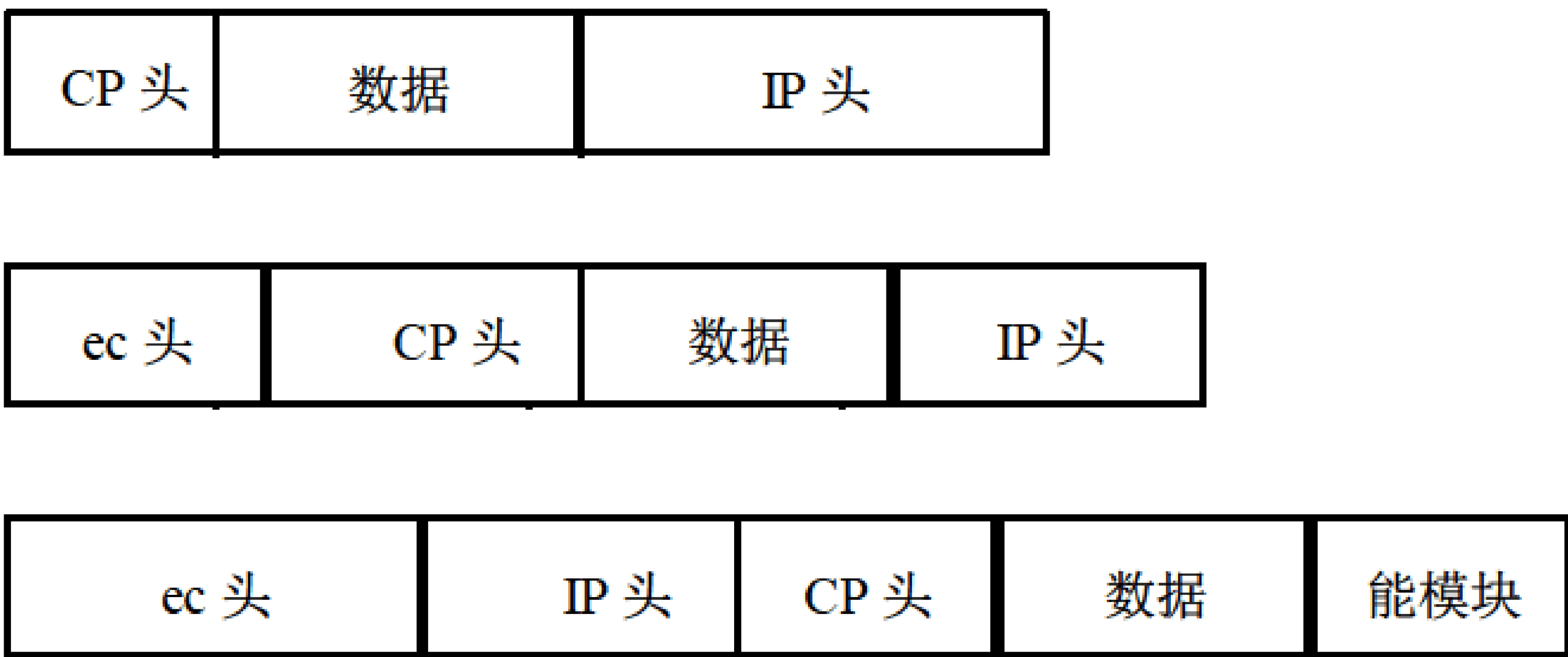


图 11-4　IPSec 数据报结构

其中，传输模式用来保护上层协议，而隧道模式用来保护整个 IP 数据报。在传输模式中，IP 头与上层协议头之间需插入一个特殊的 IPSec 头：而在隧道模式中，要保护的整个 IP 包都需封装到另一个 IP 数据包里，同时在外部与内部 IP 头之间插入一个 IPSec 头。两种 IPSec 协议（AH 和 ESP）均能同时以传输模式或隧道模式工作。由构建方法所决定，对传输模式所保护的数据包而言，其通信终点必须是一个加密的终点。在后一种情况下，通信终点便是由受保护的内部头指定的地点，而加密终点则是那些由外部 IP 头指定的地点。在 IPSec 处理结束的时候，安全网关会剥离出内部 IP 包，再将那个包转

发到它最终的目的地。

3）IPsec 协议安全性分析

IPSec 的安全性可以归纳为如下方面。

（1）当 IPSec 在路由器或防火墙中实现时，它提供很强的安全保证，可以应用于所有跨越网络边界的通信。一个实体内部的通信量不会引起与安全处理相关的开销。

（2）如果所有来自外部的通信必须使用 IP，且防火墙是 Internet 与组织的唯一入口，则 IPSec 是不能被绕过的。

（3）IPSec 位于传输层（TCP、UDP）之下，因此对应用程序是透明的。当 IPSec 在防火墙或是路由器上实现时，没有必要在用户或是服务器上更改软件。即使 IPSec 在末端系统、更高层软件（包括应用程序）上运行，也不会受到影响。

（4）IPSec 对最终用户是透明的。没有必要培训用户掌握安全机制，也没有必要基于每个用户来发行关键资料，在用户离开组织时再撤回关键资料。

如果需要，IPSec 可以为单个用户提供安全保证。这适用于站点外的工作人员，并适用于在组织内设置保密的专用子网，以用于敏感的应用程序。

2．SSL 协议

SSL 协议（Secure Socket Layer） 是 Natscape 推出的一种网络安全协议，是在传输过程通信协议（TCP/IP）上实现的一种安全协议。在 SSL 中，采用了公开密钥和私有密钥两种加密方式，它对计算机之间的整个会话进行加密。SSL 的安全服务位于 TCP 和应用层之间，可为应用层，如 HTTP、FPT、SMTP 提供安全业务，服务对象主要是 Web 应用，即客户浏览器和服务器。它的基本目标是在通信双方之间建立安全的连接，可运行在任何可靠的通信协议之上、应用层协议之下。

1）SSL 协议工作原理

在 SSL 中，所有数据被封装在记录中，记录层把从上层获得的数据分成可管理的块、可选的压缩数据、应用 MAC（Message Authentication Code）、加密、增加 SSL 首部、在 TCP 报文段中传输结果单元。被接收的数据被解密、验证、解压和重新装配，然后交付给更高级的用户。SSL 中两个重要的概念是 SSL 连接和 SSL 会话。

连接是提供恰当类型服务的传输。对于 SSL，这样的连接是点到点的关系。连接是短暂的，每个连接与一个会话相联系。连接状态由服务器和客户的随机数、服务器写 MAC 密码、客户写 MAC 密码、服务器写密钥、客户写密钥、初始化向量、序号等参数来定义。

SSL 的会话是客户和服务器之间的关联，会话通过握手协议来创建。会话定义了加密安全参数的一个集合，该集合可以被多个连接所共享。会话可以用来避免为每个连接进行昂贵的新安全参数的协商。每个会话由会话标识符、对方的证书、压缩方法、密文规约、主密钥和可重用标志等参数来定义。

2）SSL 协议工作过程

SSL 客户和服务器首次开始通信时，它们就协议版本、加密算法、是否验证及密钥等进行协商，这一过程由握手协议完成。握手过程结束后，客户端与服务器端开始交换应用层数据。握手协商过程主要包括以下几个阶段：

（1）建立安全能力

（2）服务器身份验证和密钥交换

（3）客户机验证和密钥交换

（4）完成

该阶段完成安全连接的建立。

3）SSL 协议安全性分析

（1）防止窃听及中间人攻击。

（2）防止剪贴攻击。

（3）防止重放攻击及短包攻击。

3．PGP 协议

1）PGP 协议的定义

PGP（Pretty Good Privacy）是由 Hil Zimmermann 提出的方案，是针对电子邮件在 Internet 上通信的安全问题而设计的一种混合加密系统。PGP 包含 4 个密码单元，即单钥密码（IDEA）、双钥密码（RSA）、单向杂凑算法（MD-5）和一个随机数生成算法。该协议规定公钥密码和分组密码是在同一个系统中。PGP 的用户拥有一张公钥列表，列出了所需要通信的用户及其公钥。PGP 应用程序具有很多优点，如速度快、效率高，同时具有很好的可移植性。

2）PGP 协议的加密过程

PGP 的加密过程是：先根据一些随机的环境数据（例如键盘的敲击间隔）产生一个密钥，用 IDEA 算法对明文加密。接着用接收者的 RSA 公钥对这个 IDEA 密钥进行加密，然后把这两种加密的结果作为密文发送出去。接收方接到密文后，先用自己的 RSA 私钥解密得到 IDEA 密钥，再用这个 IDEA 密钥对密文进行解密。也就是说，PGP 没有用 RSA 算法直接对明文加密，而是对 IDEA 密钥进行加密。

对于数字签名，PGP 先根据明文的内容利用 Hash 函数（散列算法）计算出一个 128 位的摘要，这个摘要就像是明文的一个精华，明文中任何改变都会导致这个精华的改变，并且从这个精华无法推导出明文的内容。发送者用自己的私钥对这个精华进行签名。因此在邮件传送过程中，任何对明文内容的改变都会导致摘要内容与签名的摘要内容不相符，以至签名的内容无效。由于 IDEA 算法的速度很快，所以不会因为邮件的数据量大而耽误时间；而 IDEA 的密钥位数较少，所以对它使用 RSA 算法在速度上也不会有太大影响。又因为 IDEA 的密码是以 RSA 加密的形式传送的，使得 PGP 既避免了 IDEA 的密钥管理缺陷，又避免了 RSA 的大量运算。PGP 的这些优点使其在邮件发送领域具有

广泛的应用。

使用 PGP 传递公钥的过程如下：假设用户 A 拥有用户 B 和用户 C 的公钥，用户 B 只拥有用户 A 的公钥，用户 C 也只拥有用户 A 的公钥。因为用户 A 和用户 B、用户 A 和用户 C 都拥有对方的公钥，所以他们之间可以安全通信。但是用户 B 和用户 C 是不能直接通信的。用户 B 和用户 C 都知道用户 A 拥有对方的公钥，如果他们都同时信任用户 A，可以从用户 A 处获得对方的公钥。即用户 A 利用自己的私钥分别对用户 B 和用户 C 的公钥签名，然后分别发给用户 B 和用户 C，这样用户 B 和用户 C 就可以安全通信了。这是一个比较简单的情况，如果用户 B 和用户 C 要经过多个用户才能获得对方的公钥，这就给用户 B 和用户 C 的正常通信带来了麻烦。同时，安全也会随着链式信任网的扩大而急速下降。

11.1.6　数据备份

1．备份的类型

随着计算机的日益普及以及信息技术的飞速发展，人们已经逐渐认识到信息安全的重要性。但是作为信息安全的重要成员——数据备份却常常被人们遗忘，这样导致的后果就是大量的有用信息被丢失，造成的后果有时是毁灭性的。

导致数据被破坏、丢失的原因很多，如硬盘的损坏、病毒的侵入等。而作为一名合格的系统管理员，关键要做到的就是保证数据的完整性以及准确性。如何才能真正做到这一点呢，这是一项非常艰巨但又非常重要的工作。一般情况下，采取的措施包括安装防火墙、杀毒软件等。但是，事情总不像人们想象的那么完美，数据的安全性和准确性一直都面临着极大的考验。因此，数据备份就显得十分有必要，同时它也是防止“主动攻击”的最重要一道防线。

数据备份包括以下几种类型，在不同的情况下，应该根据具体情况，选出最合适的方法。

（1）完全备份。是指备份全部选中的文件夹，并不依赖文件的存档属性来确定备份哪些文件（在备份过程中，任何现有的标记都被清除，每个文件都被标记为已备份。换言之，即清除存档属性）。完全备份的特点是备份所需时间最长，但恢复时间最短，操作最方便可靠。

（2）差异备份。也称差分备份，它是针对完全备份的，即备份上一次的完全备份后发生变化的所有文件。换句话说，没有发生变化的就不需要备份（差异备份过程中，只备份有标记的那些选中的文件和文件夹。它不清除标记，即备份后不标记为已备份文件。换言之，不清除存档属性）。差异备份的特点是备份时间较长，占用空间较多，但恢复时间较短。

（3）增量备份。是针对上一次备份（无论是哪种备份，这也是与差分备份不同的），即上一次备份后，所有发生变化的文件（增量备份过程中，只备份有标记的选中的文件

和文件夹，它清除标记，即备份后标记文件。换言之，清除存档属性）。增量备份的特点是备份时间较短，占用空间较少，但恢复时间较长。

（4）按需备份。也就是说，它是根据需要有选择地进行数据备份。很明显，它的特点就是有很好的选择性。

2．异地备份

数据异地备份是容灾系统的核心技术，它不同于上述介绍的备份方法，它的特点是具有异地性。它对于保证数据的一致性、可靠性及系统的可扩展性具有举足轻重的作用，通过有效的数据复制，实现远程的业务数据与本地业务数据的同步，确保一旦本地系统出现故障，远程的容灾中心能够迅速进行完整的业务接管。

异地备份在金融业中有着典型的应用，它为保证金融业的正常运行做出了巨大的贡献。在“9.11”期间，美国的金融业虽然遭受了巨大的损失，但是还能够正常运行，为什么这么巨大的灾难也没有给美国金融业带来致命的打击呢？就是因为他们对数据的异地备份做得非常好，才没有导致金融业的全面崩溃。

在进行异地备份时，要注意以下几个问题。

（1）在进行异地备份前，要集中精力进行杀毒查毒工作，避免让备份带上病毒。

（2）对于软盘，要保证磁片质量，非常有必要定期对其进行质量检查。

（3）对于 CD-RW 光盘，它的一个最大的缺点就是兼容性不好，因此最好就是由哪台刻录机刻录的盘片，就在哪台刻录机上继续刻录、改写等操作。

（4）对于移动硬盘，要做磁盘检查，保证其性能良好。

3．自动备份软件

随着人们对数据备份意识的逐渐增强，各种自动备份软件也应运而生，给我们提供了很多数据备份的选择方案，下面主要介绍几种。

1）自动备份精灵

自动备份精灵是为方便我们的备份工作而特别设计开发的软件，其最大的优点是支持网络自动备份和本机自动备份。一方面，自动备份精灵可以帮助我们定时备份数据，可以设置关机备份数据，也可以手动备份。另一方面，它也允许我们自由地选择需要备份文件的源路径和目的路径，可以查看备份日志等。

2）利用 GHOST 实现自动备份

Ghost 是最著名的硬盘复制备份工具，因为它可以将一个硬盘中的数据完全相同地复制到另一个硬盘中，因此大家就将 Ghost 这个软件称为“硬盘克隆”。Ghost 不但有硬盘到硬盘的克隆功能，还有硬盘分区、硬盘备份、系统安装、网络安装和升级系统等功能。1998 年 6 月，出品 Ghost 的 Binary 公司被著名的 Symantec 公司并购，因此该软件的后续版本就改称为 Norton Ghost，成为 Norton 系列工具软件中的一员。

3）使用 DiskWin 实现自动备份

DiskWin 主要是针对企业的备份软件。它很好地解决了企业数据备份问题。将所有

员工机的文件自动备份到服务器；管理员定义每一员工机强制备份的工作文件类型和备份时间。可规定公司不同的部门备份不同的文件类型，如公司销售部备份 Word 文件和电子邮件，公司软件开发部备份程序代码文件，设计部备份 PhotoShop 设计图片等；全盘搜索每一员工机变化的文件，保证每天新增或者变化的工作文件一个不多，一个不少，全部压缩打包，自动上传到服务器。具备多个备份的文件无论怎样重命名都只备份一个；客户端可以设置隐藏运行，无论是搜索文件还是上传备份可以不出现任何提示，对员工正常工作无任何干扰，就好像这个软件根本不存在一样。

4．几种新型的备份解决方案

对重要数据进行备份，就是为了在发生意外时能够及时进行恢复，使损失降低到最低。但是，如果备份文件存放不好，或者是备份策略不恰当，所付出的努力将付之东流。为了避免发生这种情况，我们就应该采取正确的备份方案。

一个优秀的备份解决方案应该做到以下几点。

（1）最大限度地降低对应用数据流量的影响，从而保证通信性能。

（2）最大限度地降低服务器的负载，保证服务器的性能；

（3）优化备份资源的使用，包括服务器、驱动器等。

在现代化的企业环境中，随着应用系统负载的增加，服务器的数量也在增加。但是由于磁带设备的分散特性，并且它们相互独立、不能执行全局统一的备份策略；需要的磁带机数量与应用服务器的数量成正比，所以要花费很高的维护成本。

下面介绍几种新型的备份解决方案。

1）网络备份模式

网络备份模式的原理是把一个磁带设备放置在 LAN 上，供多个服务器共享。由于网络设备模式对磁带进行统一的调度和使用，因此可以提高磁带的利用率和可管理性。需要管理的磁带驱动器的大幅度减少有助于降低成本，网络备份是一种非常好的企业备份模式。

如果普通备份的时间比较长，则可以安装一套独立的局域网，并在每套要备份的系统中连接一网卡，从而可以使备份数据与生产数据相互独立，互不影响。

在一个典型的基于 LAN 的备份模式中，生产数据和备份数据都是通过相同的 LAN 进行传输，这样需要备份的海量数据就会增加 LAN 上的流量，导致应用性能的降低。备份通常是在下班的时间进行，这样可以最大限度地减少对生产流量的影响。然而不断增长的数据量会导致备份时间的延长，而且随着企业业务的全球化，对系统的正常运行的要求也越来越高，可以用来备份的时间也越来越短。

为了在一个共同的 LAN 中消除这些潜在的冲突，可以将应用和备份隔离开来，这就是利用存储网络的方法。

另外，备份需要增加服务器的操作。服务器通常忙于处理大量对延迟和性能非常敏感的数据，数据的移动和调度需要占用额外的 CPU 周期，而进行备份通常会对应用本身

的性能造成很大的影响，因此可以采用 SCSI 扩展复制命令的备份方法加以解决。

2）用存储网络备份

这个方案是让每个应用服务器都可以通过一个专用的存储网络，直接将数据备份到某个磁带设备，而不需要经过专门的备份服务器。利用通用的共享存储设备，每个应用服务器都可以充当一个介质服务器，因为它们可以直接将数据发送到磁带。每个服务器确定一个专门的磁带驱动器，并在备份过程中独自占有该磁带驱动器。用户还可以利用对磁带库中磁带驱动器的专有访问权限对应用服务器进行配置，而不是使用共享过程。

经过存储网络传输的数据可以隔离备份数据和应用数据，从而减少 LAN 上的流量。一个磁带也可以被多个应用共享，并且可以将多个备份流量合并到所管理的磁带库和驱动器中。此时，LAN 仍可以用于在备份的服务器和客户端之间传输元数据和跟踪数据备份的状态，但是实际的备份数据将通过存储网络传输。利用网络存储备份可以隔离应用数据和备份数据，但是不能减轻服务器的 CPU 负载，因为它们仍然需要从磁带读取备份数据。

3）磁带和磁带之间直接传输数据的备份

为了减轻服务器在备份时的 CPU 负载，需要在数据不经过服务器本身的情况下，将备份数据从磁盘发送到磁带，这是通过在磁盘和磁带之间直接传输数据的机制（即 SCSI 扩展复制命令的方法）来实现的。在这种方式中，执行 SCSI 扩展复制命令的组件可能位于存储网络的交换阵列或者是服务器软件中，数据的副本会智能地从磁盘发送到磁带，而不需要经过服务器。复制并传输所要备份的数据对服务器 CPU 的负载影响非常小，这是因为服务器并不需要参与备份数据的任何具体操作，可以大大地减轻服务器的负担，保证服务器的性能不会受到备份的影响。

11.1.7 计算机病毒与免疫

1．计算机病毒

从计算机病毒刚诞生之际，它就给人们带来了麻烦，随着网络的发展，其破坏力越来越强，计算机病毒已成为危害个人系统及网络安全的一大隐患，正如生物学上的病毒能够使我们生病一样，计算机病毒会破坏计算机的正常工作。计算机病毒的一些典型破坏包括影响用户的工作（如妨碍鼠标、键盘的操作，间隔性地在用户的屏幕上显示一段文字或播放一段音乐），破坏用户系统上的一些程序（如使得 Microsoft Word 不能正常运行），大量占用系统的资源，使系统无法正常工作（蠕虫病毒的典型做法），破坏用户的数据（如删除用户的文件，格式化硬盘），有时也会破坏系统的硬件。

1）计算机病毒的定义

提到病毒，人们通常就会想到一些恶意的、时常破坏机器上的程序、数据的小程序。但如何给病毒下一个科学的、精确的定义呢？病毒的定义最早由 F.B.Cohen 于 1984 年提出，在他的经典文章 Computer Viruses-Theory and Experiments（计算机病毒一理论与实

践）中，描述如下：

“计算机病毒是这样的一种程序，它通过修改其他程序使之含有该程序本身或它的一个变体。病毒具有感染力，它可借助其使用者的权限感染他们的程序，在一个计算机系统中或网络中得以繁殖、传播。每个被感染的程序也像病毒一样可以感染其他程序，从而使更多的程序受到感染。”

2）病毒的基本特征

- 感染性
- 潜伏性
- 可触发性
- 破坏性
- 人为性
- 衍生性

3）计算机病毒的分类

分类的方式、角度是多种多样的，从病毒的工作机制角度主要分为以下 5 类。

- 引导区病毒（boot sector virus）
- 文件感染病毒（file infector virus）
- 宏病毒（Marco virus）
- 特洛伊木马（Trojan/Trojan Horse）
- 蠕虫病毒 （Worm）

2．计算机病毒免疫的原理

我们知道，计算机病毒的传染模块一般包括传染条件判断和实施传染两个部分，在病毒被激活的状态下，病毒程序通过判断传染条件的满足与否，以决定是否对目标对象进行传染。一般情况下，病毒程序在传染完一个对象后，都要给被传染对象加上传染标识，传染条件的判断就是检测被攻击对象是否存在这种标识，若存在这种标识，则病毒程序不对该对象进行传染；若不存在这种标识，则病毒程序就对该对象实施传染。由于这种原因，人们自然会想到是否能在正常对象中加上这种标识，就可以不受病毒的传染，起到免疫的作用呢？

从实现计算机病毒免疫的角度看病毒的传染，可以将病毒的传染分成两种。第一种是像香港病毒、1575 病毒这样，在传染前先检查待传染的扇区或程序里是否含有病毒代码，如果没有找到则进行传染，如果找到了则不再进行传染。这种用作判断是否为病毒自身的病毒代码被称作传染标志或免疫标志。第二种是在传染时不判断是否存在免疫标志，病毒只要找到一个可传染对象就进行一次传染。就像黑色星期五那样，一个文件可能被黑色星期五反复传染多次，滚雪球一样越滚越大（需要说明的是，黑色星期五病毒的程序中具有判别传染标志的代码，由于程序设计错误，使判断失败，形成现在的情况，

对文件会反复感染，传染标志形同虚设）。

目前常用的免疫方法有如下两种。

1）针对某一种病毒进行的计算机病毒免疫

例如对小球病毒，在 DOS 引导扇区的 1FCH 处填上 1357H，小球病毒一旦检查到这个标志就不再对它进行传染了。对于 1575 文件型病毒，免疫标志是文件尾的内容为 0CH 和 0AH 的两个字节，1575 病毒若发现文件尾含有这两个字节，则不进行传染。这种方法的优点是可以有效地防止某一种特定病毒的传染。但缺点很严重，主要有以下几点。

（1）对于没有设感染标识的病毒不能达到免疫的目的。有的病毒只要在激活的状态下， 会无条件的把病毒传染给被攻击对象，而不论这种对象是否已经被感染过或者是否具有某种标识。

（2）当出现这种病毒的变种不再使用这个免疫标志时或出现新病毒时，免疫标志发挥不了作用。

（3）某些病毒的免疫标志不容易仿制，非要加上这种标志不可，则对原来的文件要做大的改动。例如对大麻病毒就不容易做免疫标志。

（4）由于病毒的种类较多，又由于技术上的原因，不可能对一个对象加上各种病毒的免疫标识，这就使得该对象不能对所有的病毒具有免疫作用。

（5）这种方法能阻止传染，却不能阻止病毒的破坏行为，仍然放任病毒驻留在内存中。目前使用这种免疫方法的商品化反病毒软件已不多见了。

2）基于自我完整性检查的计算机病毒的免疫方法

目前这种方法只能用于文件而不能用于引导扇区。这种方法的原理是：为可执行程序增加一个免疫外壳，同时在免疫外壳中记录有关用于恢复自身的信息。免疫外壳占 1～3KB。执行具有这种免疫功能的程序时，免疫外壳首先得到运行，检查自身的程序大小、校验生成日期和时间等情况，没有发现异常时才转去执行受保护的程序。

但是，它仍存在如下一些缺点和不足。

（1）每个受到保护的文件都要增加 1～3KB，需要额外的存储空间。

（2）现在使用中的一些校验码算法不能满足防病毒的需要，被某些种类的病毒感染的文件不能被检查出来。

（3）无法对付覆盖方式的文件型病毒。

（4）有些类型的文件不能使用外加免疫外壳的防护方法，这样将使那些文件不能正常执行。

当某些尚不能被病毒检测软件检查出来的病毒感染了文件，而该文件又被免疫外壳包在里面时，这个病毒就像穿了“保护盔甲”，使查毒软件查不到它，而它却能在得到运行机会时跑出来继续传染扩散。

11.2　信息安全管理和评估

11.2.1　安全管理技术

由于数据在网络上进行传输时，可能会存在各种攻击，因此，必须加强对网络安全的管理。概括性地说，安全管理技术就是监督、组织和控制网络通信服务以及信息处理所必需的各种技术手段和措施的总称。其目标是确保计算机网络的持续正常运行，并在计算机网络运行出现异常时能及时响应和排除故障。

1．安全管理的发展现状

在 20 世纪 90 年代中后期，随着因特网的发展以及社会信息化程度越来越高，各种安全设备在网络中的应用也越来越多，市场上开始出现了独立的安全管理产品。

相对而言，国外计算机网络安全管理的需求多样，起步较早，已经形成了较大规模的市场，有一部分产品逐渐在市场上获得了用户的认可。近年来，国内厂商也开始推出网络安全管理产品，但一般受技术实力限制，大多是针对自己的安全设备开发的集中管理软件、安全审计系统等。

由于各种网络安全产品的作用体现在网络中的不同方面，统一的网络安全管理平台必然要求对网络中部署的安全设备进行协同管理，这是统一安全管理平台的最高追求目标。

2．网络安全管理技术简介

安全管理（Security Management，SM），不管是对于个人管理，还是对企业管理（Enterprise Management），都是十分重要的。从信息管理的角度看，安全管理涉及到策略与规程、安全缺陷以及保护所需的资源、防火墙、密码加密问题、鉴别与授权、客户机/服务器认证系统、报文传输安全以及对病毒攻击的保护等。

实际上，安全管理不是一个简单的软件系统，它包括的内容非常多，主要涵盖了安全设备的管理、安全策略管理、安全风险控制和安全审计等几个方面。

（1）安全设备管理：是指对网络中所有的安全产品，如防火墙、VPN、防病毒、入侵检测（网络、主机）和漏洞扫描等产品实现统一管理、统一监控。

（2）安全策略管理：是指管理、保护及自动分发全局性的安全策略，包括对安全设备、操作系统及应用系统的安全策略的管理。

（3）安全分析控制：确定、控制并消除或缩减系统资源的不定事件的总过程，包括风险分析、选择、实现与测试、安全评估及所有的安全检查（含系统补丁程序检查）。

（4）安全审计：对网络中的安全设备、操作系统及应用系统的日志信息收集汇总，实现对这些信息的查询和统计；并通过对这些集中信息的进一步分析，可以得出更深层次的安全分析结果。

3. 安全管理主要解决以下问题

1）集中化的安全策略管理（Centralized Security Policy Management，CSPM）

企业的安全保障需要自上而下地制定安全策略，这些安全策略会被传送并装配到不同的执行点（Enforcement Point）中。

2）实时安全监视（Real-Time Security Awareness，RTSA）

企业用户实时了解企业网络内的安全状况。

3）安全联动机制（Contain Mechanism，CM）

安全设备之间需要具备有中心控制或无中心控制的安全联动机制，即当 IDS 发现在某网段有入侵动作时，它需要通知防火墙阻断此攻击。

4）配置与补丁管理（Configuration and Patching Management）

企业用户可以通过对已发现的安全缺陷快速反应，大大提高自己抵抗风险的能力。

5）统一的权限管理（Privilege Management across the Enterprise）

通过完善的权限管理和身份认证实现对网络资源使用的有效控制和审计。

11.2.2 安全性规章

1. 信息系统安全制度

一段时间以来，国际和国内一些著名网站被“黑”的事件引起了社会多方的关注。计算机信息系统的安全问题越来越受到重视，因为安全问题将影响到电子商务、国家信息甚至是国防等各个方面。针对大幅度上升的黑客攻击、病毒传播和有害信息传播等计算机违法犯罪活动，有关部门出台了一系列的信息系统安全法规与制度，从而进一步保证了信息系统的安全运行。

1）计算机信息系统安全保护等级划分标准

《计算机信息系统安全保护等级划分标准》规定，从 2001 年 1 月 1 日起对计算机信息系统安全保护实行等级划分，此举标志着我国计算机信息系统安全保护纳入了等级管理的轨道。

由公安部提出并组织制定、国家质量技术监督局发布的强制性国家标准《计算机信息系统安全保护等级划分准则》，将计算机信息系统的安全保护等级划分为用户自主保护级、系统审计保护级、安全标记保护级、结构化保护级和访问验证保护级 5 个级别。用户可以根据自己计算机信息系统的重要程度确定相应的安全保护级别，并针对相应级别进行建设。

实行安全等级保护制度后，公安机关能够通过规范、科学、公正的评定和监督管理，全面、及时地预防和发现计算机信息系统建设和使用中存在的安全风险和安全漏洞，有利于提高公安机关对计算机信息系统安全保护的监督管理水平。此外，实行这一制度还有利于提高信息安全产业化发展水平，为安全产品的普及使用提供广阔的市场和发展空间。

2）计算机信息安全保护条例

根据公安部的有关规定，计算机信息系统安全保护包括以下几个方面。

（1）实体安全：包括周围危险建筑与设施、监控系统、防火措施、防水措施、机房环境、防雷措施、备用电源、防静电措施、用电质量和防盗措施等。

（2）网络通信安全：包括通信设备的场所标志、重要的通信线路及通信控制装置备份、加密措施、网络运行状态安全审计跟踪措施、网络系统访问控制措施和工作站身份识别措施等。

（3）软件与信息安全：包括操作系统及数据库访问控制措施、应用软件、系统信息能防止恶意攻击和非法存取、数据库及系统状态监控、防护措施、用户身份识别措施、系统用户信息异地备份等。

（4）管理组织与制度安全：包括专门的安全防范组织和计算机安全员、健全的安全管理规章制度、详尽的工作手册和完整的工作记录、定期进行风险分析，制定灾难处理对策、建立安全培训制度、制定人员的安全管理制度等。

（5）安全技术措施：包括灾难恢复的技术措施、开发工作与业务工作分离的措施、应用业务、系统安全审计功能、系统操作日志、服务器备份措施、计算机防病毒措施等。

2．计算机防毒制度

为了加强计算机病毒的防治管理工作，2000 年公安部发布了《计算机病毒防治管理办法》，规定各级公安机关负责本行政区域内的计算机病毒防治管理工作。

规定禁止制作、传播计算机病毒，向社会发布虚假计算机病毒疫情，承担计算机病毒的认定工作的机构应由公安部公共信息网络安全监察部门批准，计算机信息系统的使用单位应当履行防治计算机病毒的职责。

11.3　信息安全保障体系

对一个信息网络，必须从总体上规划，建立一个科学全面的信息安全保障体系，从而实现信息系统的整体安全。一个全面的信息安全保障体系，应该能够解决信息系统存在的大部分安全威胁。目前的信息安全威胁主要有：针对系统稳定性和可靠性的破坏行为，包括从外部网络针对内部网络的攻击入侵行为和病毒破坏等；大量信息设备的使用、维护和管理问题，包括违反规定的计算机、打印机和其他信息基础设施的滥用，以及信息系统违规使用软件和硬件的行为；知识产权和内部机密材料等有价值信息存储、使用和传输的保密性、完整性和可靠性存在可能的威胁，其中尤其以信息的保密性存在威胁的可能性最大。

针对这些复杂且技术手段各异的信息安全威胁，要建立一个完整的信息安全保障体系，包含以下几个方面的内容。

1）建立统一的身份认证体系

身份认证是信息交换最基础的要素，如果不能确认交换双方的实体身份，那么信息的安全就根本无从得到保证。身份认证的含义是广泛的，其泛指一切实体的身份，包括人、计算机、设备和应用程序等，只有确认了所有这些信息在存储、使用和传输中可能涉及的实体，信息的安全性才有可能得到基本保证。

2）建立统一的信息安全管理体系

建立对所有信息实体有效的信息安全管理体系，对信息网络系统中的所有计算机、输出端口、存储设备、网络、应用程序和其他设备进行有效集中的管理，从而有效管理和控制信息网络中存在的安全风险。信息安全管理体系的建立主要集中在技术性系统的建立上，同时，也应该建立相应的管理制度，才能使信息安全管理系统得到有效实施。

3）建立规范的信息安全保密体系

信息的保密性是一个大型信息应用网络不可缺少的需求，所以，必须建立符合规范的信息安全保密体系。这个体系不仅仅应该提供完善的技术解决方案，也应该建立相应的信息保密管理制度。

4）建立完善的网络边界防护体系

重要的信息网络一般会跟公共的互联网进行一定程度的分离，在内部信息网络和互联网之间存在一个网络边界。必须建立完善的网络边界防护体系，使得内部网络既能够与外部网络进行信息交流，同时也能防止从外网发起的对内部网络的攻击等安全威胁。

此外，要加快信息安全立法，建立信息安全法制体系，这样才能做到有法可依、有法必依。建立国家信息安全组织管理体系，加强国家信息安全机构及职能；建立高效能的、职责分工明确的行政管理和业务组织体系；建立信息安全标准和评估体系；建立国家信息安全技术保障体系，使用科学技术实施安全的防护保障。

第 12 章　系统安全架构设计

12.1　信息系统安全架构的简单描述

信息安全的特征是为了保证信息的机密性、完整性、可用性、可控性和不可抵赖性。信息系统的安全保障是以风险和策略为基础，在信息系统的整个生命周期中提供包括技术、管理、人员和工程过程的整体安全，在信息系统中保障信息的这些安全特征，并实现组织机构的使命。许多信息系统的用户需要提供一种方法和内容对信息系统的技术框架、工程过程能力和管理能力提出安全性要求，并进行可比性的评估、设计和实施。

12.1.1　信息安全的现状及其威胁

随着社会信息化进程的加快，计算机及网络已经在各行各业中得到了广泛的应用，同时一些重要单位如政府机关、部队、企业财务和人事部门已经越来越依赖于计算机。毫无疑问，在不远的将来，计算机和网络的普及程度会比现在有更大的提高，这种普及将会产生两方面的效应：其一，各行各业的业务运转几乎完全依赖于计算机和网络，各种重要数据如政府文件、工资档案、财务账目和人事资料等将全部依托计算机和网络存储、传输；其二，大多数人对计算机的了解更加全面，有更多的计算机技术水平较高的人可以采用种种手段对信息资源进行攻击。目前，信息安全主要可能会受到的威胁可以总结为以下几个方面，如图 12-1 所示。

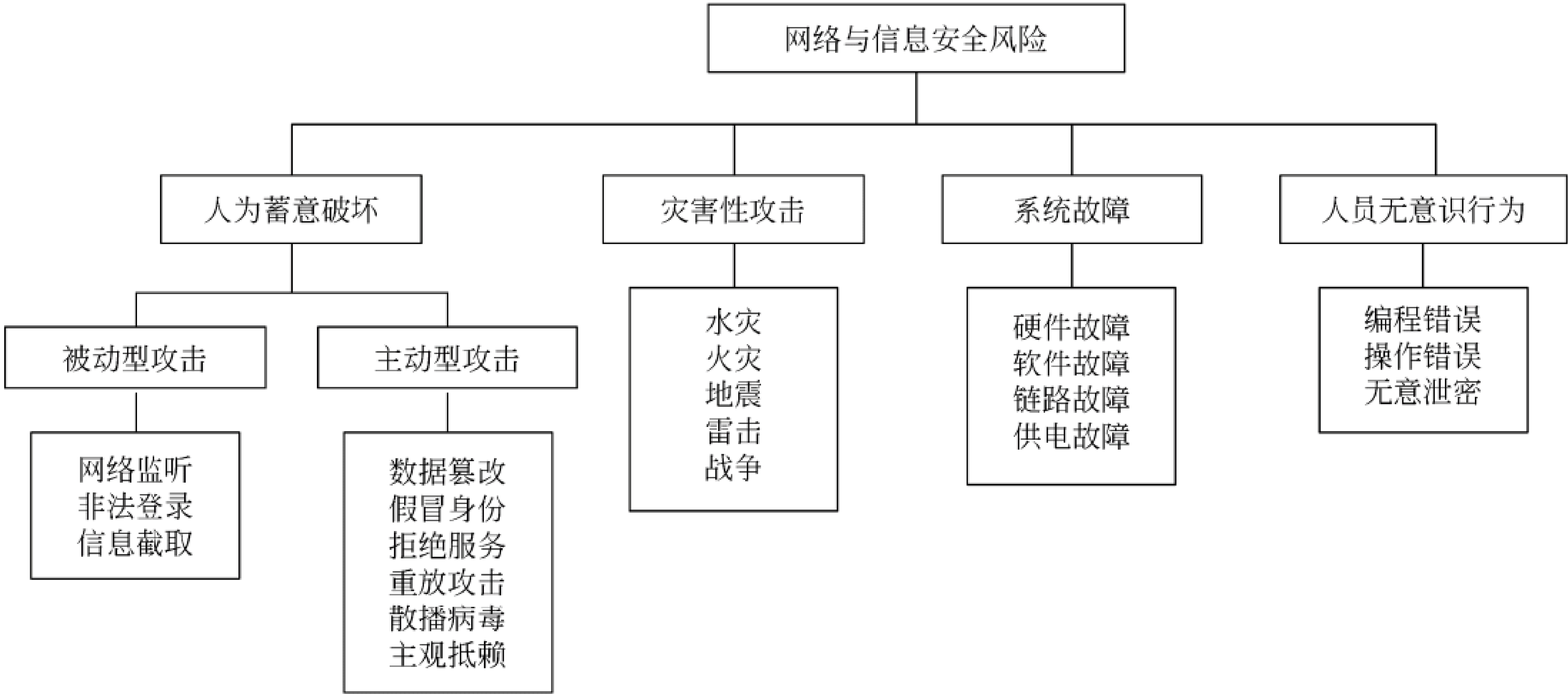


图 12-1　网络与信息安全风险

对于信息系统来说，威胁可以是针对物理环境、通信链路、网络系统、操作系统、应用系统以及管理系统等方面。物理安全威胁，是指对系统所用设备的威胁，自然灾害、电源故障、操作系统引导失败或数据库信息丢失、设备被盗/被毁造成数据丢失或信息泄露。通信链路安全威胁，是指在传输线路上安装窃听装置或对通信链路进行干扰。网络安全威胁，互联网的开放性、国际性与无安全管理性，对内部网络形成严重的安全威胁。操作系统安全威胁，对系统平台最危险的威胁是在系统软件或硬件芯片中植入威胁，如“木马”和“陷阱门”、BIOS 有万能密码。应用系统安全威胁，是指对于网络服务或用户业务系统安全的威胁，也受到“木马”和“陷阱门”的威胁。管理系统安全威胁，必须从人员管理上杜绝安全漏洞。

具体来讲，常见的安全威胁有如下几种。

（1）信息泄露：信息被泄露或透露给某个非授权的实体。

（2）破坏信息的完整性：数据被非授权地进行增删、修改或破坏而受到损失。

（3）拒绝服务：对信息或其他资源的合法访问被无条件地阻止。

（4）非法使用（非授权访问）：某一资源被某个非授权的人、或以非授权的方式使用。

（5）窃听：用各种可能的合法或非法的手段窃取系统中的信息资源和敏感信息。例如对通信线路中传输的信号进行搭线监听，或者利用通信设备在工作过程中产生的电磁泄露截取有用信息等。

（6）业务流分析：通过对系统进行长期监听，利用统计分析方法对诸如通信频度、通信的信息流向、通信总量的变化等参数进行研究，从而发现有价值的信息和规律。

（7）假冒：通过欺骗通信系统（或用户）达到非法用户冒充成为合法用户，或者特权小的用户冒充成为特权大的用户的目的。黑客大多是采用假冒进行攻击。

（8）旁路控制：攻击者利用系统的安全缺陷或安全性上的脆弱之处获得非授权的权利或特权。例如，攻击者通过各种攻击手段发现原本应保密，但是却又暴露出来的一些系统“特性”。利用这些“特性”，攻击者可以绕过防线守卫者侵入系统的内部。

（9）授权侵犯：被授权以某一目的使用某一系统或资源的某个人，却将此权限用于其他非授权的目的，也称作“内部攻击”。

（10）特洛伊木马：软件中含有一个察觉不出的或者无害的程序段，当它被执行时，会破坏用户的安全。这种应用程序称为特洛伊木马（Trojan Horse）。

（11）陷阱门：在某个系统或某个部件中设置了“机关”，使得当提供特定的输入数据时，允许违反安全策略。

（12）抵赖：这是一种来自用户的攻击，例如，否认自己曾经发布过的某条消息、伪造一份对方来信等。

（13）重放：所截获的某次合法的通信数据备份，出于非法的目的而被重新发送。

（14）计算机病毒：所谓计算机病毒，是一种在计算机系统运行过程中能够实现传

染和侵害的功能程序。一种病毒通常含有两个功能：一种功能是对其他程序产生“感染”；另外一种或者是引发损坏功能、或者是一种植入攻击的能力。

（15）人员不慎：一个授权的人为了钱或利益、或由于粗心，将信息泄露给一个非授权的人。

（16）媒体废弃：信息被从废弃的磁盘或打印过的存储介质中获得。

（17）物理侵入：侵入者通过绕过物理控制而获得对系统的访问。

（18）窃取：重要的安全物品，如令牌或身份卡被盗。

（19）业务欺骗：某一伪系统或系统部件欺骗合法的用户或系统自愿地放弃敏感信息。

通过对网络面临的安全风险威胁和实施相应控制措施的支出进行合理的评价，提出有效合理的安全技术，形成提升网络信息的安全性质的安全方案，是安全架构设计的根本目标。在实际应用中，可以从安全技术的角度提取出 5 个方面的内容：认证鉴别、访问控制、内容安全、冗余恢复和审计响应。

12.1.2　国内外影响较大的标准和组织

1．标准

1）国外的标准

有如下标准。

（1）可信计算机标准评估规则橘皮书（TCSEC，美国）

（2）欧洲 ITSEC 标准

（3）加拿大 CTCPEC 标准

（4）美国联邦准则（FC）

（5）美国信息技术安全评价通用准则（CC）

（6）ISO 安全体系结构标准（ISO7498-2-1989）<信息处理系统开放系统互连基本参考模型第二部分安全体系结构>

（7）美国国家安全局：信息保障技术框架（IATF）

2）我国的标准

（1）主管部门：公安部、信息产业部和国家技术标准局等

（2）主要技术标准如下。

- GA163-1997（计算机信息系统安全专用产品分类原则）
- GB17895-1999（计算机信息系统安全保护等级划分准则）
- GB/T9387.2-1995（信息处理系统开放系统互连基本参考模型第二部分安全体系结构）
- GB 15834.1-1995（信息处理数据加密实体鉴别机制第一部分：一般模型）
- GB 4943-1995（信息技术设备的安全）

2．组织

1）国际标准化组织（ISO）

ISO 的信息技术标准化委员会 TC97 在 1984 年 1 月，专门组织了一个分技术委员会 SC20，负责制定数据加密技术的国际标准；之后在 1987 年，ISO 的 TC97 和 IEC 的 TCs47B/83 合并成为 ISO/IEC 联合技术委员会（JTC1）；1990 年 4 月，ISO 将原来的数据加密分技术委员会 SC20，更名为安全技术分技术委员会 SC27，专门从事信息技术安全一般方法和技术的标准化工作。而 ISO/TC68 负责银行业务应用范围内有关信息安全标准的制定，它主要制定行业应用标准，在组织上和标准之间与 SC27 有着密切的联系。

由于信息技术的发展，开放系统互连的网络体系结构的广泛应用，信息技术安全标准化越来越受到人们的重视。在信息技术安全分委会的成立会上，研究了信息技术安全标准化的发展规划，明确了指导思想，确定了工作目标，制定了实施计划，提出了具体的措施，正在为建立完整的信息技术安全标准体系而积极组织开展研究工作和标准制定工作。

2）信息技术安全分技术委员会

1984 年 7 月，在我国的全国计算机与信息处理标准化技术委员会下，建立了相应的数据加密分技术委员会，在国家技术监督局和原电子工业部的领导下，归口国内外的信息技术数据加密的标准化工作。随着信息技术的发展和工作范围的扩大，在原数据加密分委员会的基础上，于 1997 年 8 月改组成了信息技术安全分技术委员会（与 ISO/IEC JTC1/SC27 信息技术的安全技术分委会对应）。它是一个具有广泛代表性、权威性和军民结合的信息安全标准化组织。其工作范围是负责信息和通信安全的通用框架、方法、技术和机制的标准化，归口管理国内外对应的标准化工作。其技术安全包括开放式安全体系结构、各种安全信息交换的语义规则、在有关的应用程序接口和协议引用安全功能接口等。

12.2 系统安全体系架构规划框架及其方法

安全技术体系架构是对组织机构信息技术系统的安全体系结构的整体描述。安全技术体系架构能力是拥有信息技术系统的组织机构根据其策略的要求和风险评估的结果，参考相关技术体系构架的标准和最佳实践，结合组织机构信息技术系统的具体现状和需求，建立的符合组织机构信息技术系统战略发展规划的信息技术系统整体体系框架；它是组织机构信息技术系统战略管理的具体体现。技术体系架构能力是组织机构执行安全技术整体能力的体现，它反映了组织机构在执行信息安全技术体系框架管理达到预定的成本、功能和质量目标上的度量。

安全技术体系架构过程的目标是建立可持续改进的安全技术体系架构的能力，信息技术系统千变万化，有各种各样的分类方式，为从技术角度建立一个通用的对象分析模型，在本书中将信息系统抽象成一个基本完备的信息系统分析模型，如图 12-2 所示。从信息技术系统分析模型出发，建立整个信息技术系统的安全架构。

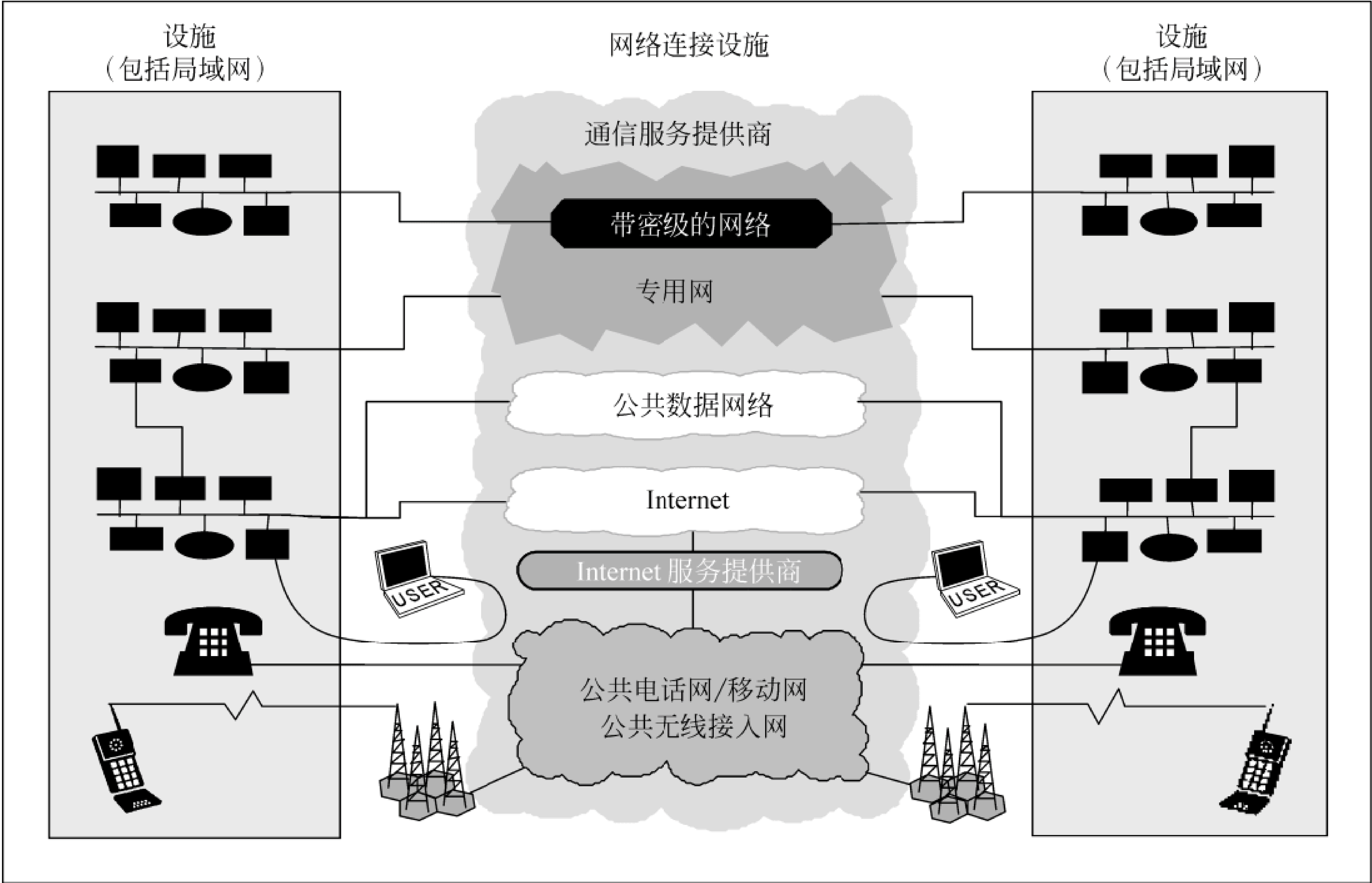


图 12-2　信息技术系统分析模型

一般来说，OSI 参考模型将网络划分为物理、数据链路、网络、传输、会话、表示和应用 7 层，Andrew S.Tanenbau 综合 OSI 参考模型和 TCP/IP 参考模型将网络划分为物理、数据链路、网络、传输、应用 5 层。在本模型中，首先需要做的就是对网络结构层次进行划分，考虑到安全评估是以安全风险威胁分析入手的，而且在实际的网络安全评估中会发现，主机和存储系统占据了大量的评估考察工作，虽然主机和存储系统都属于应用层，但本模型由于其重要性，特将其单列为一个层次，因此根据网络中风险威胁的存在实体划分出 5 个层次的实体对象：应用、存储、主机、网络和物理。

信息系统安全规划是一个非常细致和非常重要的工作，首先需要对企业信息化发展的历史情况进行深入和全面的调研，知道家底、掌握情况，针对信息系统安全的主要内容进行整体的发展规划工作。下面用图 12-3 表示信息系统安全体系的框架。

从图 12-3 可以看出，信息系统安全体系主要是由技术体系、组织机构体系和管理体系三部分共同构成的。技术体系是全面提供信息系统安全保护的技术保障系统，该体系由物理安全技术和系统安全技术两大类构成。组织体系是信息系统的组织保障系统，由机构、岗位和人事三个模块构成。机构分为领导决策层、日常管理层和具体执行层；岗位是信息系统安全管理部门根据系统安全需要设定的负责某一个或某几个安全事务的职位；人事是根据管理机构设定的岗位，对岗位上在职、待职和离职的员工进行素质教育、业绩考核和安全监管的机构。管理体系由法律管理、制度管理和培训管理三部分组成。

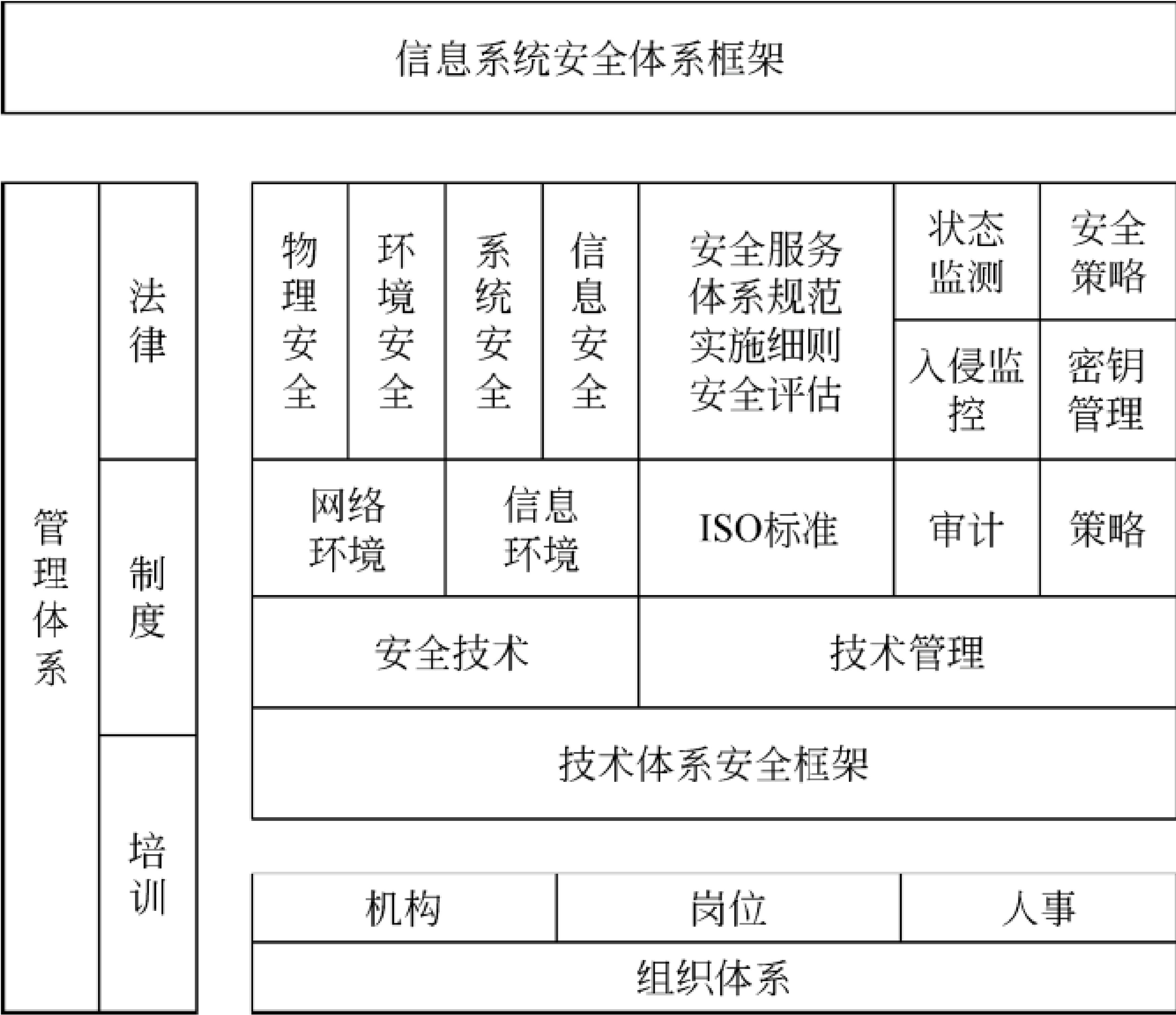


图 12-3　信息系统安全体系

信息系统安全体系清楚了之后，就可以针对以上描述的内容进行全面的规划。信息系统安全规划的层次方法与步骤可以有不同，但是规划内容与层次应该是相同。规划的具体环节、相互之间的关系和具体方法如图 12-4 所示。

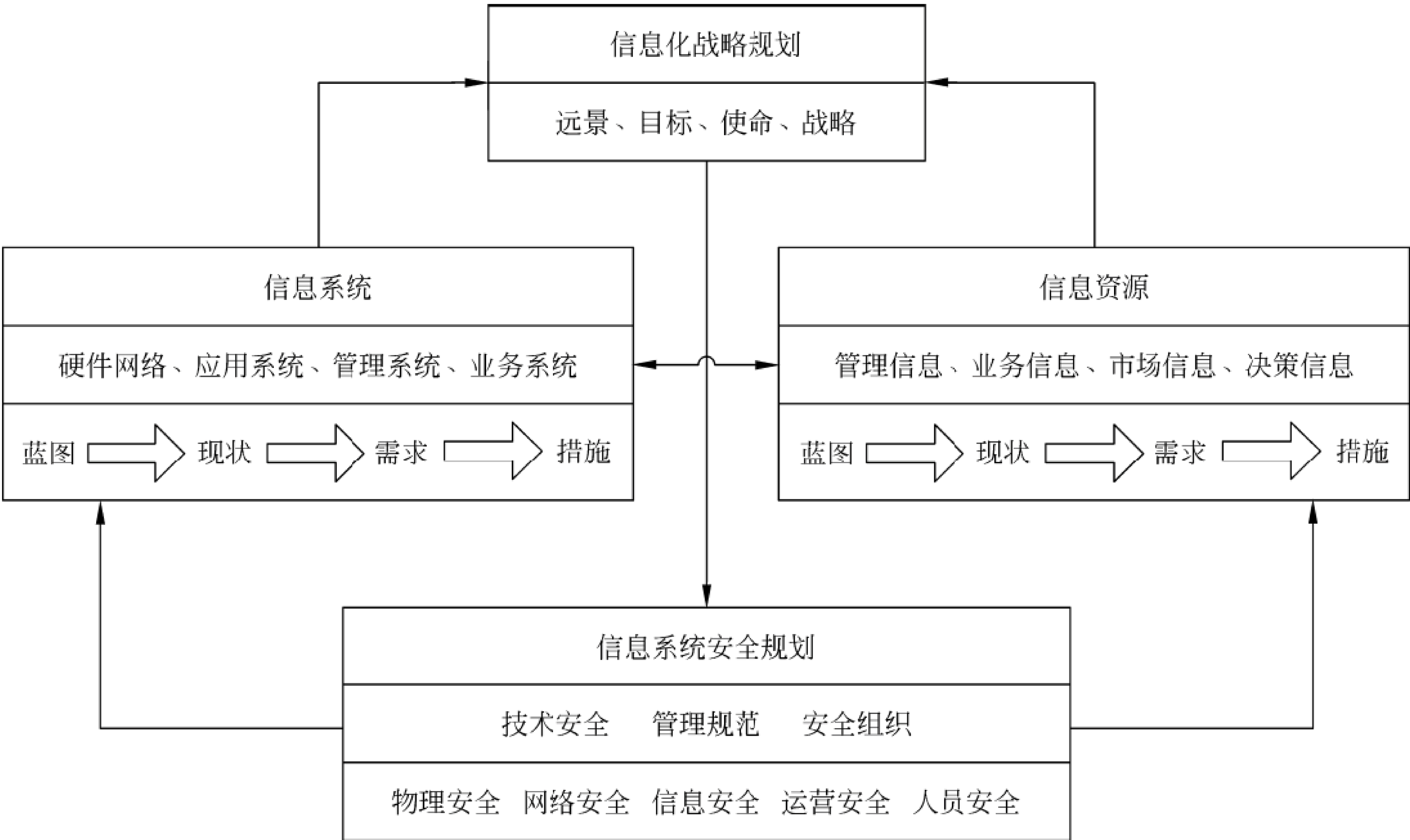


图 12-4　信息系统安全规划框架图

1．信息系统安全规划依托企业信息化战略规划

信息化战略规划是以整个企业的发展目标、发展战略和企业各部门的业务需求为基础，结合行业信息化方面的需求分析、环境分析和对信息技术发展趋势的掌握，定义出企业信息化建设的远景、使命、目标和战略，规划出企业信息化建设的未来架构，为信息化建设的实施提供一副完整的蓝图，全面系统地指导企业信息化建设的进程。信息系统安全规划依托企业信息化战略规划，对信息化战略的实施起到保驾护航的作用。信息系统安全规划的目标应该与企业信息化的目标是一致的，而且应该比企业信息化的目标更具体明确、更贴近安全。信息系统安全规划的一切论述都要围绕着这个目标展开和部署。

2．信息系统安全规划需要围绕技术安全、管理安全、组织安全考虑

信息系统安全规划的方法可以不同、侧重点可以不同，但都需要围绕技术安全、管理安全、组织安全进行全面的考虑。规划的内容基本上应该涵盖：确定信息系统安全的任务、目标、战略以及战略部门和战略人员，并在此基础上制定出物理安全、网络安全、系统安全、运营安全、人员安全的信息系统安全的总体规划。物理安全包括环境设备安全、信息设备安全、网络设备安全、信息资产设备的物理分布安全等。网络安全包括网络拓扑结构安全、网络的物理线路安全、网络访问安全（防火墙、入侵检测系统和 VPN 等）等。系统安全包括操作系统安全、应用软件安全和应用策略安全等。运营安全应在控制层面和管理层面保障，包括备份与恢复系统安全、入侵检测功能、加密认证功能、漏洞检查及系统补丁功能、口令管理等。人员安全包括安全管理的组织机构、人员安全教育与意识机制、人员招聘及离职管理、第三方人员安全管理等。

3．信息系统安全规划以信息系统与信息资源的安全保护为核心

信息系统安全规划的最终效果应该体现在对信息系统与信息资源的安全保护上，因此规划工作需要围绕着信息系统与信息资源的开发、利用和保护工作进行，要包括蓝图、现状、需求和措施 4 个方面。

（1）对信息系统与信息资源的规划需要从信息化建设的蓝图入手，知道企业信息化发展策略的总体目标和各阶段的实施目标，制定出信息系统安全的发展目标。

（2）对企业的信息化工作现状进行整体的、综合、全面的分析，找出过去工作中的优势与不足。

（3）根据信息化建设的目标提出未来几年的需求，这个需求最好可以分解成若干个小的方面，以便于今后的落实与实施。

（4）要写明在实施工作阶段的具体措施与办法，提高规划工作的执行力度。信息系统安全规划服务于企业信息化战略目标，信息系统安全规划做得好，企业信息化工作的实现就有了保障。信息系统安全规划是企业信息化发展战略的基础性工作，不是可有可无而是非常重要。由于企业信息化的任务与目标不同，所以信息系统安全规划包括的内容就不同，建设的规模就有很大的差异，因此信息系统安全规划无法从专业书籍或研究

资料中找到非常有针对性的、有帮助的适用法则，也不可能给出一个规范化的信息系统安全规划的模板。在这里提出信息系统安全规划框架与方法，给出了信息系统安全规划工作的一种建设原则、建设内容、建设思路，具体规划还需要深入细致地进行本地化的调查与研究。

12.3 网络安全体系架构设计

介绍信息系统安全体系的目的，就是将普遍性安全原理与信息系统的实际相结合，形成满足信息系统安全需求的安全体系结构。

12.3.1 OSI 的安全体系架构概述

国家标准《信息处理系统工程开放系统互联基本参考模型—第二部分：安全体系结构》（GB/T 9387.2 -1995）（等同于 ISO 7498-2），以及互联网安全体系结构（RFC 2401），是两个普遍适用的安全体系结构，目的在于保证开放系统进程与进程之间远距离安全交换信息。这些标准在参考模型的框架内，建立起一些指导原则与约束条件，从而提供了解决开放互联系统中安全问题的一致性方法。

OSI 安全体系结构提供以下内容。

（1）提供安全服务与有关安全机制在体系结构下的一般描述，这些服务和机制必须是为体系结构所配备的。

（2）确定体系结构内部可以提供这些服务的位置。

（3）保证安全服务完全准确地得以配置，并且在信息系统的安全周期中一直维持，安全功能务必达到一定强度的要求。

国家标准《信息处理系统工程开放系统互联基本参考模型—第二部分：安全体系结构》（GB/T 9387.2 -1995）（等同于 ISO 7498-2），给出了基于 OSI 参考模型的 7 层协议之上的信息安全体系结构。其核心内容是：为了保证异构计算机进程与进程之间远距离交换信息的安全，它定义了该系统 5 大类安全服务，以及提供这些服务的 8 类安全机制及相应的 OSI 安全管理，并可根据具体系统适当地配置于 OSI 模型的 7 层协议中。图 12-5 所示的三维安全空间解释了这一体系结构。

在 OSI 7 层协议中除第 5 层（会话层）外，每一层均能提供相应的安全服务。实际上，最适合配置安全服务的是在物理层、网络层、运输层及应用层上，其他层都不宜配置安全服务。

ISO 开放系统互联安全体系的 5 类安全服务包括鉴别、访问控制、数据机密性、数据完整性和抗抵赖性。

分层多点安全技术体系架构，也称为深度防御安全技术体系架构，它通过以下方式将防御能力分布至整个信息系统中。

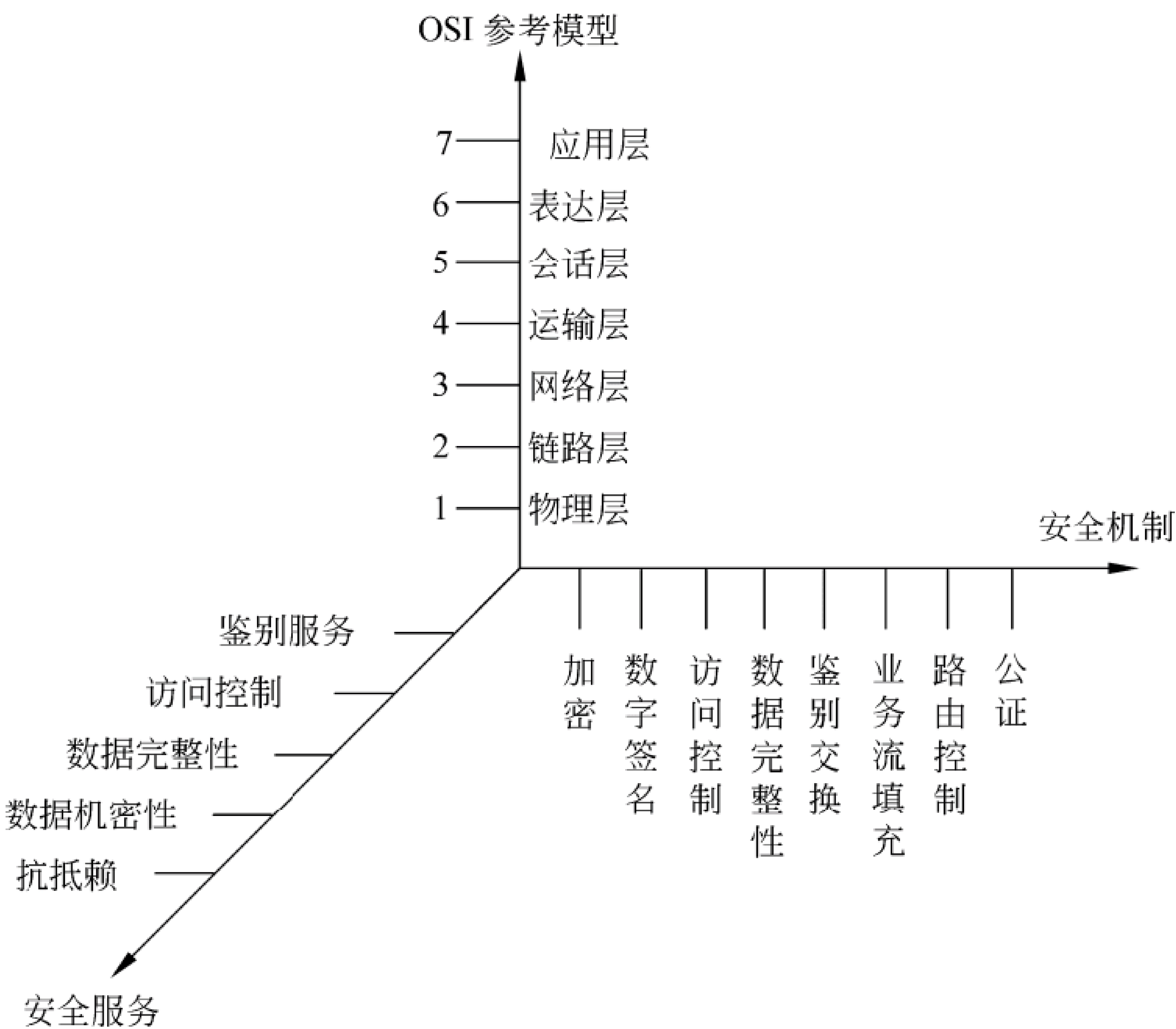


图 12-5　信息安全体系结构示意图

（1）多点技术防御。在对手可以从内部或外部多点攻击一个目标的前提下，多点技术防御通过对以下多个防御核心区域的防御达到抵御所有方式的攻击的目的。

① 网络和基础设施。为了确保可用性，局域网和广域网需要进行保护以抵抗各种攻击，如拒绝服务攻击等。为了确保机密性和完整性，需要保护在这些网络上传送的信息以及流量的特征以防止非故意的泄露。

② 边界。为了抵御主动的网络攻击，边界需要提供更强的边界防御，例如流量过滤和控制以及入侵检测。

③ 计算环境。为了抵御内部、近距离的分布攻击，主机和工作站需要提供足够的访问控制。

（2）分层技术防御。即使最好的可得到的信息保障产品也有弱点，其最终结果将使对手能找到一个可探查的脆弱性，一个有效的措施是在对手和目标间使用多个防御机制。为了减少这些攻击成功的可能性和对成功攻击的可承担性，每种机制应代表一种唯一的障碍并同时包括保护和检测方法。例如，在外部和内部边界同时使用嵌套的防火墙并配合以入侵检测就是分层技术防御的一个实例。

支撑性基础设施为网络、边界和计算环境中信息保障机制运行基础的支撑性基础设施，包括公钥基础设施以及检测和响应基础设施。

（1）公钥基础设施。提供一种通用的联合处理方式，以便安全地创建、分发和管理公钥证书和传统的对称密钥，使它们能够为网络、边界和计算环境提供安全服务。这些

服务能够对发送者和接收者的完整性进行可靠验证，并可以避免在未获授权的情况下泄露和更改信息。公钥基础设施必须支持受控的互操作性，并与各用户团体所建立的安全策略保持一致。

（2）检测和响应基础设施。检测和响应基础设施能够迅速检测并响应入侵行为。它也提供便于结合其他相关事件观察某个事件的“汇总”性能。另外，它也允许分析员识别潜在的行为模式或新的发展趋势。

必须提醒的是，信息系统的安全保障不仅仅依赖于技术，还需要集成的技术和非技术防御手段。一个可接受级别的信息保障依赖于人员、管理、技术和过程的综合。

图 12-6 描述了分层多点安全技术体系架构。

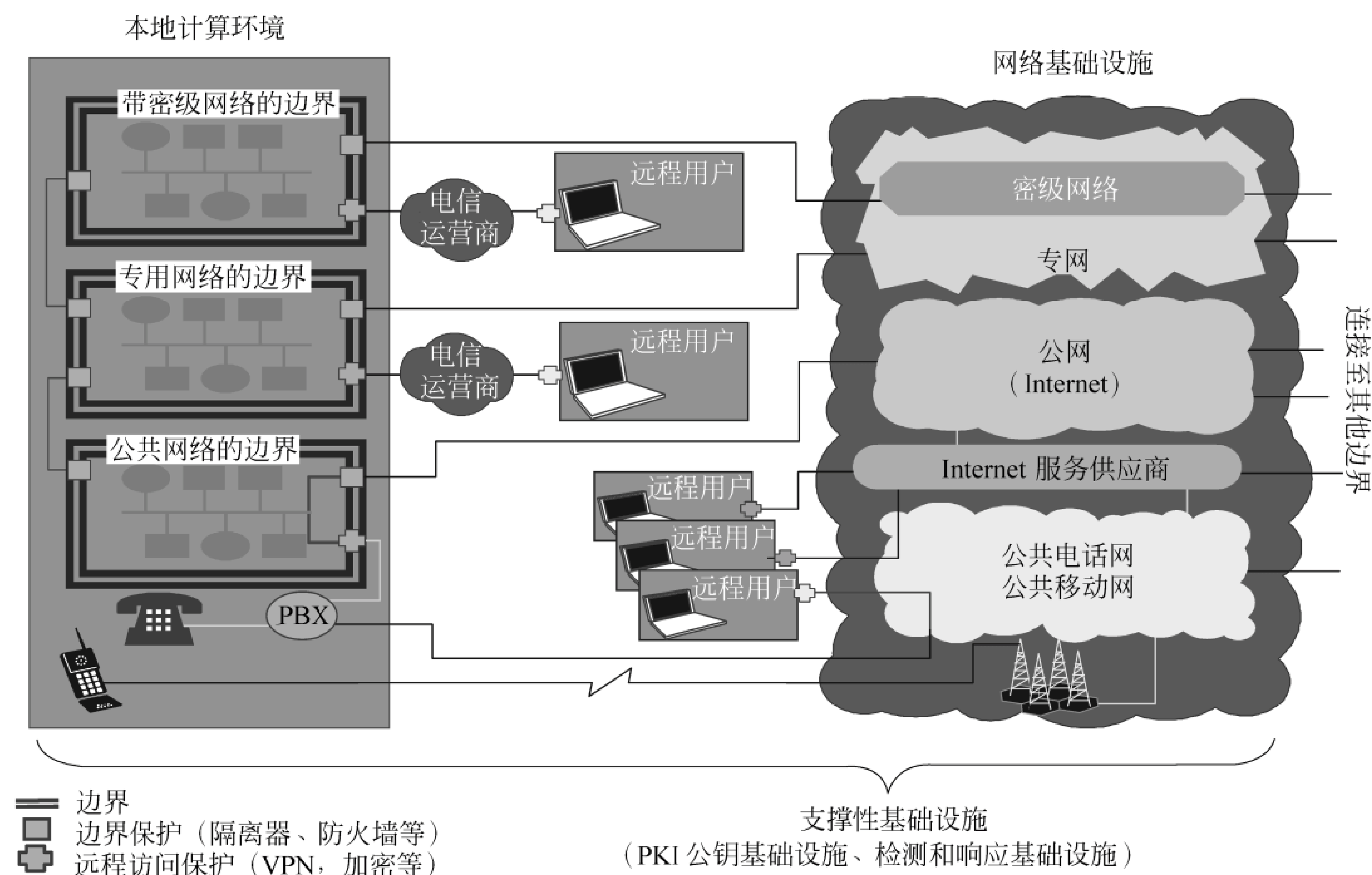


图 12-6　分层多点安全技术体系架构

分层多点安全技术体系架构为信息系统安全保障提供了框架和进一步分析所需的重点区域划分。在具体的技术方案实践中，应从使命和需求的实际情况出发制定适合组织机构要求的技术体系和方案。

12.3.2　鉴别框架

鉴别（Authentication）的基本目的，就是防止其他实体占用和独立操作被鉴别实体的身份。鉴别提供了实体声称其身份的保证，只有在主体和验证者的关系背景下，鉴别

才是有意义的。鉴别有两种重要的关系背景：一是实体由申请者来代表，申请者与验证者之间存在着特定的通信关系（如实体鉴别）；二是实体为验证者提供数据项来源。

鉴别的方式主要基于以下 5 种。

（1）已知的，如一个秘密的口令。

（2）拥有的，如 IC 卡、令牌等。

（3）不改变的特性，如生物特征。

（4）相信可靠的第三方建立的鉴别（递推）。

（5）环境（如主机地址等）。

鉴别信息（Artificial Intelligence，AI）是指申请者要求鉴别到鉴别过程结束所生成、使用和交换的信息。鉴别信息的类型有交换鉴别信息（交换 AI）、申请鉴别信息（申请 AI）和验证鉴别信息（验证 AI）。

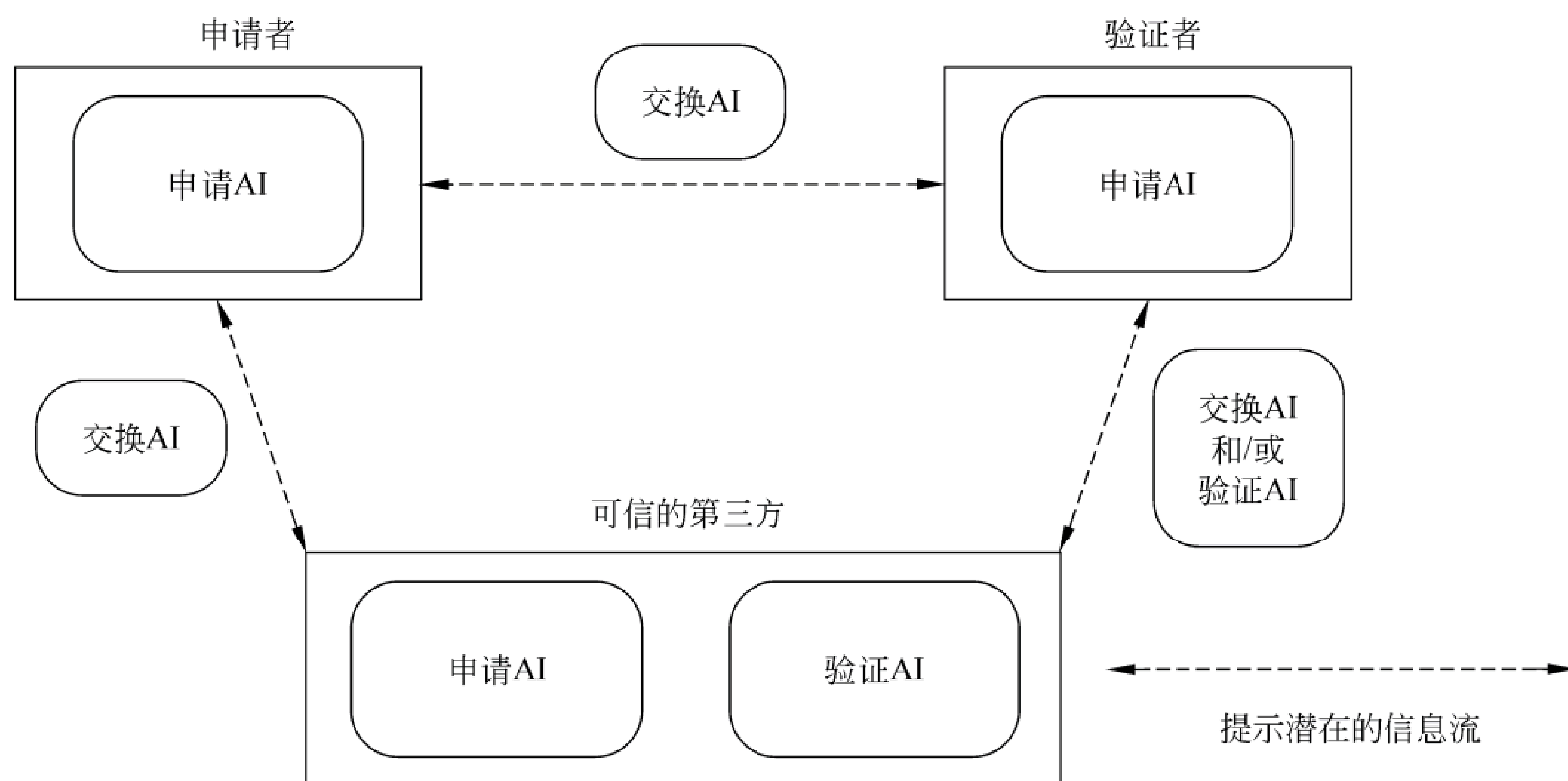


图 12-7　申请者、验证者、可信第三方之间的关系及三种鉴别信息类型

在某些情况下，为了产生交换 AI，申请者需要与可信第三方进行交互。类似的，为了验证交换 AI，验证者也需要同可信第三方进行交互。在这种情况下，可信第三方持有相关实体的验证 AI，也可能使用可信第三方来传递交换 AI。实体也可能需要持有鉴别可信第三方中所使用的鉴别信息。

鉴别服务分为以下阶段：安装阶段；修改鉴别信息阶段；分发阶段；获取阶段；传送阶段；验证阶段；停活阶段；重新激活阶段；取消安装阶段。

在安装阶段，定义申请 AI 和验证 AI。修改鉴别信息阶段，实体或管理者申请 AI 和验证 AI 变更（如修改口令）。在分发阶段，为了验证交换 AI，把验证 AI 分发到各实

体（如申请者或验证者）以供使用。在获取阶段，申请者或验证者可得到为鉴别实例生成特定交换 AI 所需的信息，通过与可信第三方进行交互或鉴别实体间的信息交换可得到交换 AI。例如，当使用联机密钥分配中心时，申请者或验证者可从密钥分配中心得到一些信息，如鉴别证书。在传送阶段，在申请者与验证者之间传送交换 AI。在验证阶段，用验证 AI 核对交换 AI。在停活阶段，将建立一种状态，使得以前能被鉴别的实体暂时不能被鉴别。在重新激活阶段，使在停活阶段建立的状态将被终止。在取消安装阶段，实体从实体集合中被拆除。

12.3.3 访问控制框架

访问控制（Access Control）决定开放系统环境中允许使用哪些资源、在什么地方适合阻止未授权访问的过程。在访问控制实例中，访问可以是对一个系统（即对一个系统通信部分的一个实体）或对一个系统内部进行的。

图 12-8 和图 12-9 说明了访问控制的基础性功能。

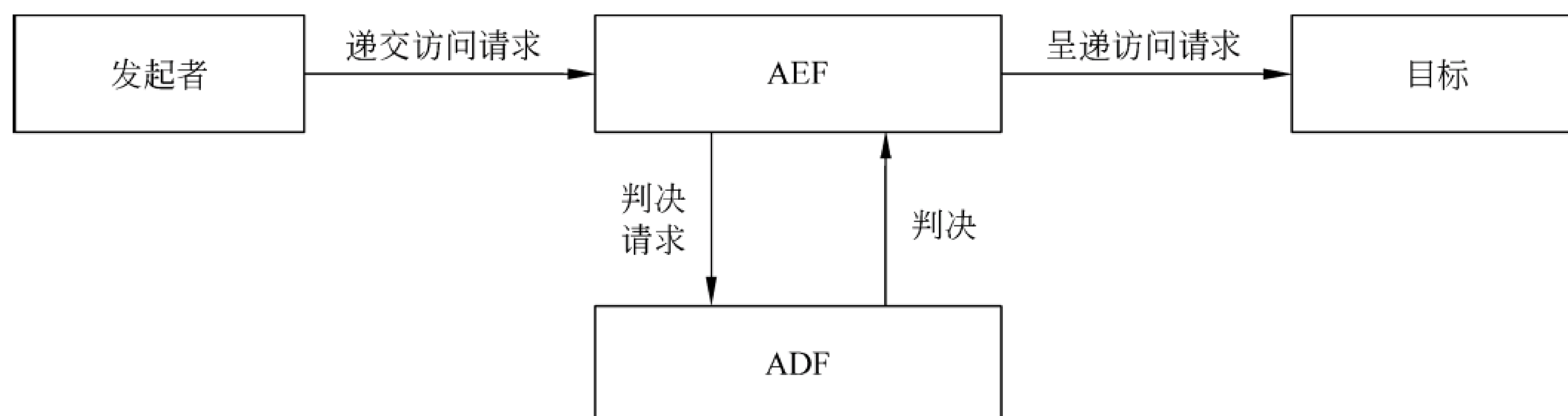


图 12-8　基本访问控制功能示意图

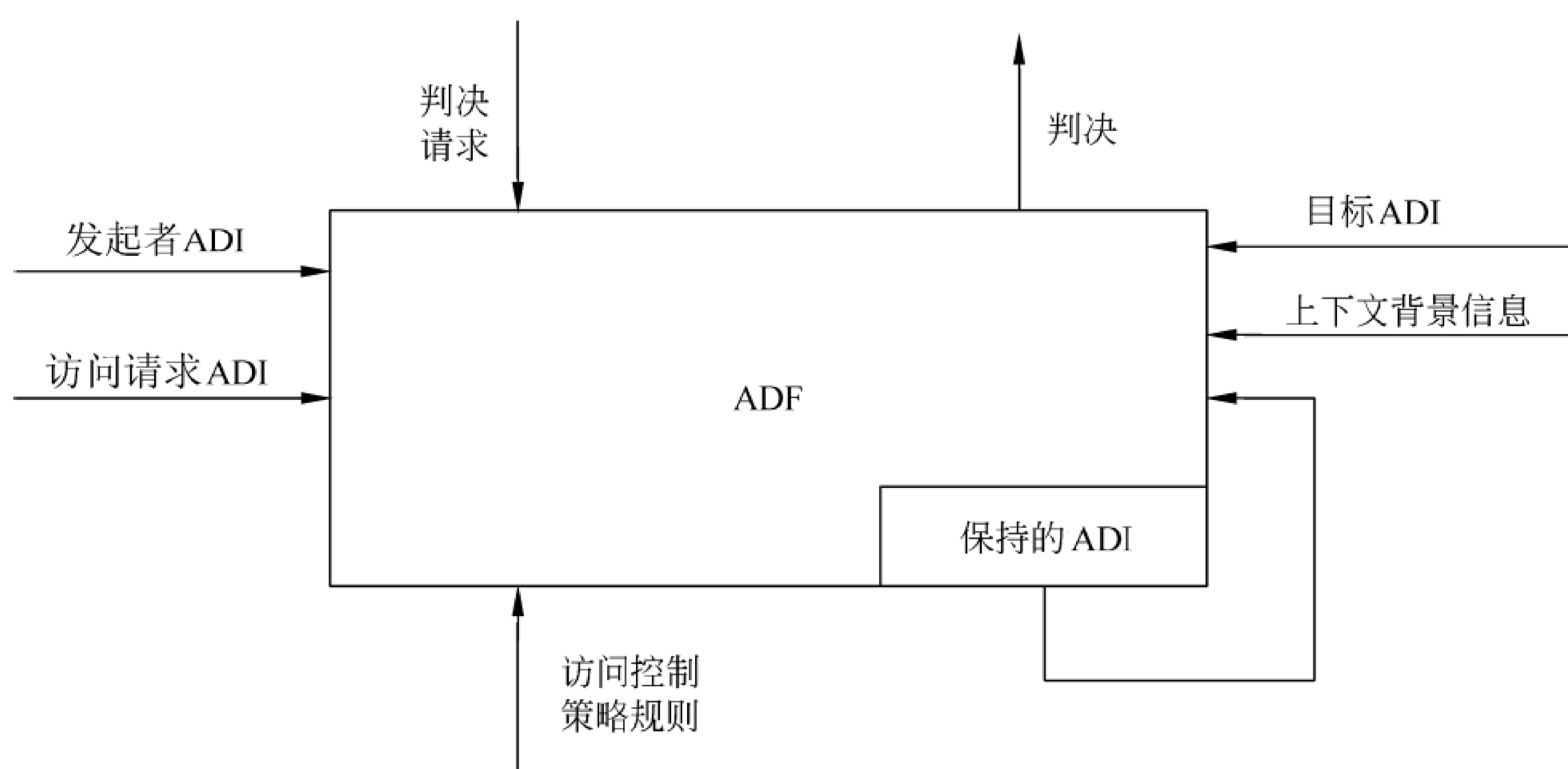


图 12-9　ADF 示意图

ACI（访问控制信息）是用于访问控制目的的任何信息，其中包括上下文信息。ADI（访问控制判决信息）是在做出一个特定的访问控制判决时可供 ADF 使用的部分（或全部）ACI。ADF（访问控制判决功能）是一种特定功能，它通过对访问请求、ADI 以及该访问请求的上下文使用访问控制策略规则而做出访问控制判决。AEF（访问控制实施功能）确保只有对目标允许的访问才由发起者执行。

涉及访问控制的有发起者、AEF、ADF 和目标。发起者代表访问或试图访问目标的人和基于计算机的实体。 目标代表被试图访问或由发起者访问的，基于计算机或通信的实体。例如，目标可能是 OSI 实体、文件或者系统。访问请求代表构成试图访问部分的操作和操作数。

当发起者请求对目标进行特殊访问时，AEF 就通知 ADF 需要一个判决来做出决定。为了作出判决，给 ADF 提供了访问请求（作为判决请求的一部分）和下列几种访问控制判决信息（ADI）。

（1）发起者 ADI（ADI 由绑定到发起者的 ACI 导出）。

（2）目标 ADI（ADI 由绑定到目标的 ACI 导出）。

（3）访问请求 ADI（ADI 由绑定到访问请求的 ACI 导出）。

ADF 的其他输入是访问控制策略规则（来自 ADF 的安全域权威机构）和用于解释 ADI 或策略的必要上下文信息。上下文信息包括发起者的位置、访问时间或使用中的特殊通信路径。基于这些输入，以及可能还有以前判决中保留下来的 ADI 信息，ADF 可以做出允许或禁止发起者试图对目标进行访问的判决。该判决传递给 AEF，然后 AEF 允许将访问请求传给目标或采取其他合适的行动。

在许多情况下，由发起者对目标的逐次访问请求是相关的。应用中的一个典型例子是在打开与同层目标的连接应用进程后，试图用相同（保留）的 ADI 执行几个访问。对一些随后通过连接进行通信的访问请求，可能需要给 ADF 提供附加的 ADI 以允许访问请求。在另一些情况中，安全策略可能要求对一个或多个发起者与一个或多个目标之间的某种相关访问请求进行限制。这时，ADF 可能使用与多个发起者和目标有关的先前判决中所保留的 ADI 来对特殊访问请求作出判决。

如果得到 AEF 的允许，访问请求只涉及发起者与目标的单一交互。尽管发起者和目标之间的一些访问请求是完全与其他访问请求无关的，但常常是两个实体进入一个相关的访问请求集合中，如质询应答模式。在这种情况下，实体根据需要同时或交替地变更发起者和目标角色，可以由分离的 AEF 组件、ADF 组件和访问控制策略对每一个访问请求执行访问控制功能。

12.3.4　机密性框架

机密性（Confidentiality）服务的目的是确保信息仅仅是对被授权者可用。由于信息是通过数据表示的，而且数据可能导致关系的变化（如文件操作可能导致目录改变或可

用存储区域的改变），因此信息能通过许多不同的方式从数据中导出。例如，通过理解数据的含义（如数据的值）导出；通过使用数据相关的属性（如存在性、创建的数据、数据大小、最后一次更新的日期等）进行推导；通过研究数据的上下文关系，即那些与之相关的其他数据实体导出；通过观察数据表达式的动态变化导出。

信息的保护通过确保数据被限制于授权者获得，或通过特定方式表示数据来获得，这种保护方式的语义是，数据只对那些拥有某种关键信息的人才是可访问的。有效的机密性保护要求必要的控制信息（如密钥和 RCI 等）是受到保护的，这种保护机制和用来保护数据的机制是不同的（如密钥可以通过物理手段保护等）。

在机密性框架中用到被保护的环境和被交叠保护的环境两个概念。在被保护环境中的数据，通过使用特别的安全机制（或多个机制）保护。在一个被保护环境中的所有数据以类似方法受到保护。当两个或更多的环境交叠的时候，交叠中的数据能被多重保护。可以推断，从一个环境移到另一个环境的数据的连续保护必然涉及到交叠保护环境。

机密性机制

数据的机密性可以依赖于所驻留和传输的媒体。因此，存储数据的机密性能通过使用隐藏数据语义（如加密）或将数据分片的机制来保证。数据在传输中的机密性能通过禁止访问的机制、通过隐藏数据语义的机制或通过分散数据的机制得以保证（如跳频等）。这些机制类型能被单独使用或者组合使用。

1）通过禁止访问提供机密性

通过禁止访问的机密性能通过在 ITU-T Rec. 812 或 ISO/IEC 10181-3 中描述的访问控制获得，以及通过物理媒体保护和路由选择控制获得。通过物理媒体保护的机密性保护可以采取物理方法保证媒体中的数据只能通过特殊的有限设备才能检测到。数据机密性通过确保只有授权的实体才能使这些机制本身有效的方式来实现。通过路由选择控制的机密性保护机制的目的，是防止被传输数据项表示的信息未授权泄露。在这一机制下只有可信和安全的设施才能路由数据，以达到支持机密性服务的目的。

2）通过加密提供机密性

这些机制的目的是防止数据泄露在传输或存储中。加密机制分为基于对称的加密机制和基于非对称加密的机密机制。

除了以下两种机密性机制外，还可以通过数据填充、通过虚假事件（如把在不可信链路上交换的信息流总量隐藏起来）、通过保护 PDU 头和通过时间可变域提供机密性。

12.3.5 完整性框架

完整性（Integrity）框架的目的是通过阻止威胁或探测威胁，保护可能遭到不同方式危害的数据完整性和数据相关属性完整性。所谓完整性，就是数据不以未经授权方式进行改变或损毁的特征。

完整性服务有几种分类方式：根据防范的违规分类违规操作分为未授权的数据修

改、未授权的数据创建、未授权的数据删除、未授权的数据插入和未授权的数据重放。依据提供的保护方法分为阻止完整性损坏和检测完整性损坏。依据是否包括恢复机制分为具有恢复机制的和不具有恢复机制的。

完整性机制的类型

由于保护数据的能力与正在使用的媒体有关。完整性机制是有区别的，包括如下类型。

（1）阻止对媒体访问的机制。包括物理隔离的、不受干扰的信道；路由控制；访问控制。

（2）用以探测对数据或数据项序列的非授权修改的机制。未授权修改包括未授权数据创建、数据删除以及数据重复。而相应的完整性机制包括密封、数字签名、数据重复（作为对抗其他类型违规的手段）、与密码变换相结合的数字指纹和消息序列号。

按照保护强度，完整性机制可分为不作保护；对修改和创建的探测；对修改、创建、删除和重复的探测；对修改和创建的探测并带恢复功能；对修改、创建、删除和重复的探测并带恢复功能。

12.3.6　抗抵赖框架

抗抵赖（Non-repudiation）服务包括证据的生成、验证和记录，以及在解决纠纷时随即进行的证据恢复和再次验证。

框架所描述的抗抵赖服务的目的是提供有关特定事件或行为的证据。事件或行为本身以外的其他实体可以请求抗抵赖服务。抗抵赖服务可以保护的行为实例有发送 X.400 消息；在数据库中插入记录、请求远程操作等。

当涉及消息内容的抗抵赖服务时，为提供原发证明，必须确认数据原发者身份和数据完整性。为提供递交证明，必须确认接收者身份和数据完整性。在某些情况下，还可能需要涉及上下文关系（如日期、时间、原发者 / 接收者的地点等）的证据。

抗抵赖服务提供下列可在试图抵赖的事件中使用的设备：证据生成、证据记录、验证生成的证据、证据的恢复和重验。

纠纷可以在纠纷两方之间直接通过检查证据解决。但是，纠纷也可能不得不通过仲裁者解决，该仲裁者评估并确定是否发生过有纠纷的行为或事件。

抗抵赖由 4 个独立的阶段组成：证据生成；证据传输、存储和恢复；证据验证；解决纠纷，如图 12-10 所示。

1）证据生成

在这个阶段中，证据生成请求者请求证据生成者为事件或行为生成证据。卷入事件或行为中的实体，称为证据实体，其卷入关系由证据建立。根据抗抵赖服务的类型，证据可由证据实体、或可能与可信第三方的服务一起生成、或者单独由可信第三方生成。

2）证据传输、存储和恢复

在这个阶段，证据在实体间传输或从存储器取出来或传到存储器。

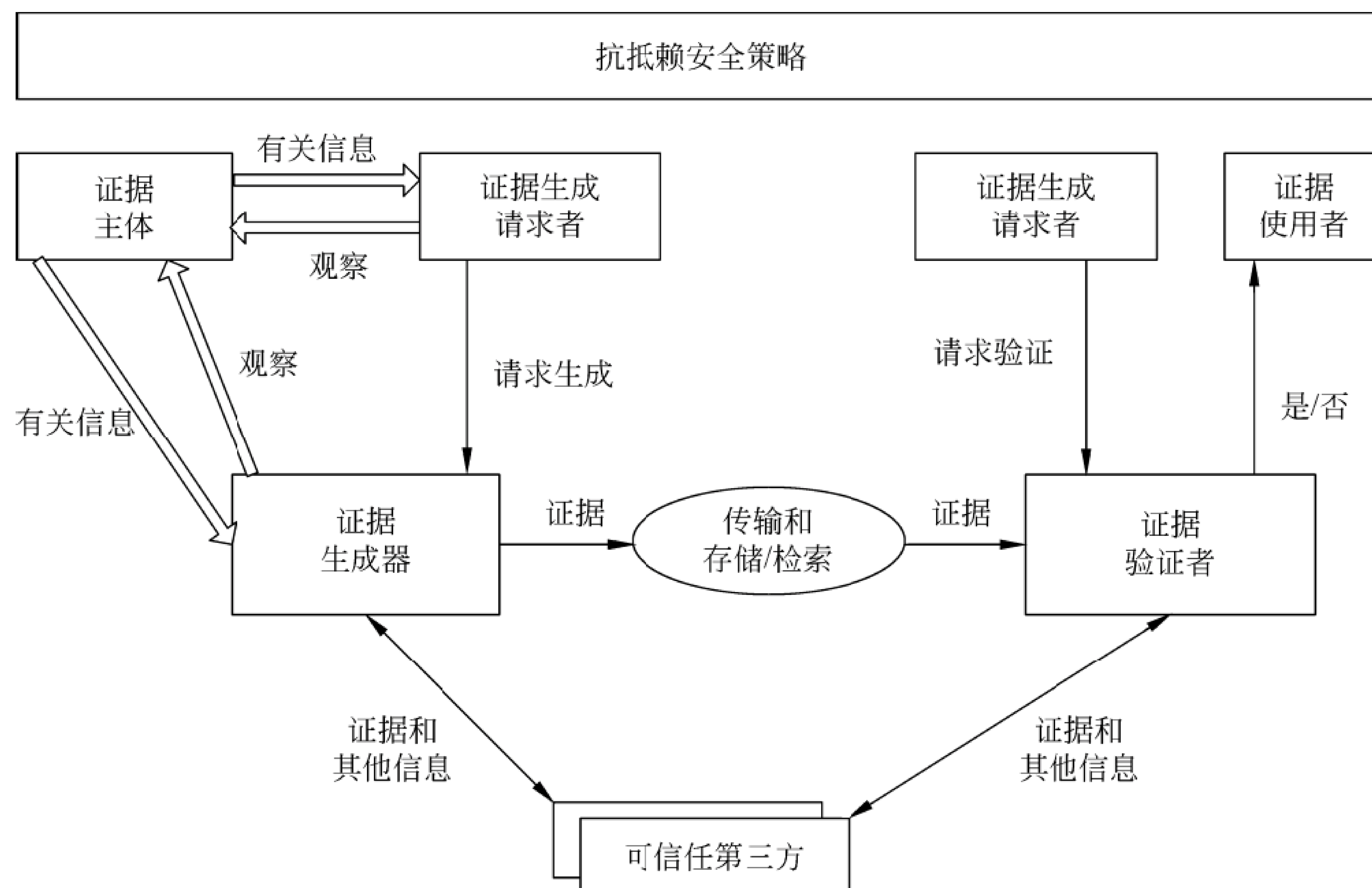


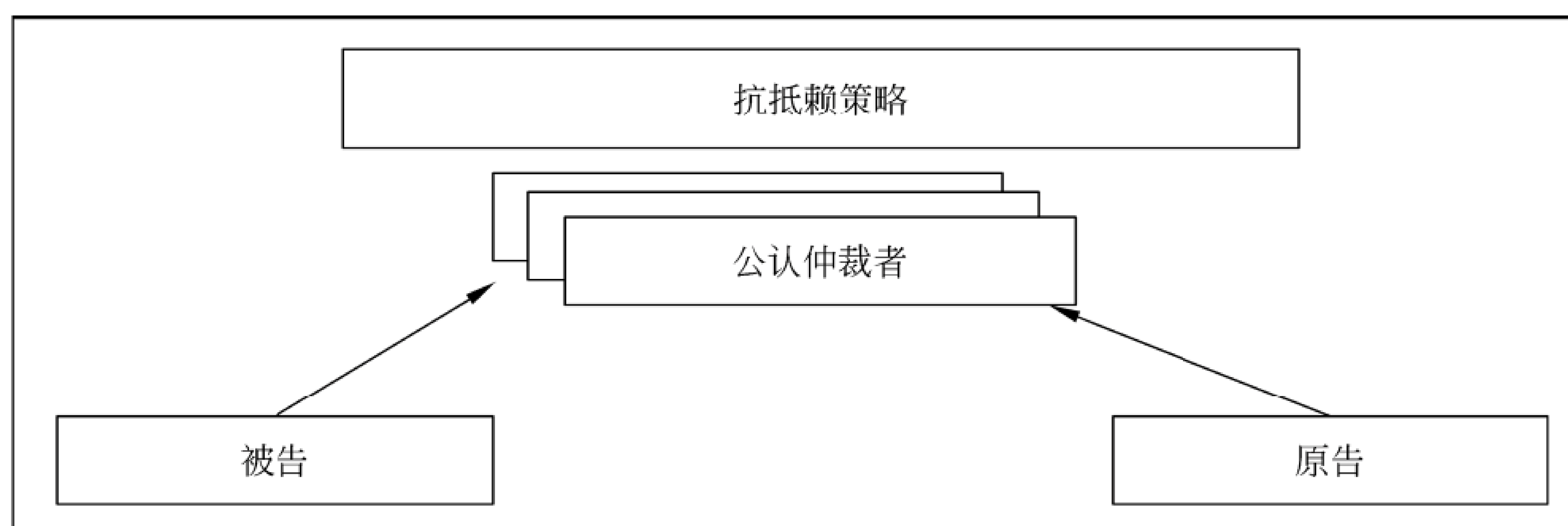
图 12-10 参与生成、传输、存储/恢复和证实阶段的实体

3）证据验证

在这个阶段，证据在证据使用者的请求下被证据验证者验证。本阶段的目的是在出现纠纷的事件中，让证据使用者确信被提供的证据确实是充分的。可信第三方服务也可参与，以提供验证该证据的信息。

4）解决纠纷

在解决纠纷阶段，仲裁者有解决双方纠纷的责任。图 12-11 描述了纠纷解决阶段。



注：本图是示意图，并非定义

图 12-11 抗抵赖过程的纠纷解决阶段

12.4　数据库系统的安全设计

在数据库系统中，由于数据的集中管理，随之而来的是多用户存取，以及近年来跨网络的分布系统的急速发展。数据库的安全问题可以说已经构成信息系统最为关键的环节，而电子政务中所涉及的数据库密级更高、实时性更强。因此，有必要根据其特殊性完善安全策略，这些安全策略应该能保证数据库中的数据不会被有意地攻击或无意地破坏。不会发生数据的外泄、丢失和毁损，即实现了数据库系统安全的完整性、保密性和可用性。从数据库管理系统的角度而言，要采取的安全策略一般为用户管理、存取控制、数据加密、审计跟踪和攻击检测，从而解决数据库系统的运行安全和信息安全。

下面分别从数据库安全设计的评估标准和完整性设计两方面进行讨论。

12.4.1　数据库安全设计的评估标准

随着人们对安全问题的认识和对安全产品的要求不断提高，在计算机安全技术方面逐步建立了一套安全评估标准，以规范和指导安全信息的建立、安全产品的生产，并能较准确地评测产品的安全性能指标。在当前各国制定和采用的标准中，最重要的是 1985 年美国国防部颁布的“可信计算机系统评估标准（Trusted Computer System Evaluation Criteria，TCSEC）”桔皮书（简称为 DoD85）。1991 年，美国国家计算机安全中心（The National Computer Seaurity Center，NCSC）又颁布了“可信计算机评估标准关于可信数据库管理系统的解释（Trusted Database Interpretation，TDI）”。我国也于 1994 年 2 月发布了“中华人民共和国计算机信息系统安全保护条例”。在 TCSEC 中，将安全系统分为 4 大类 7 个等级。

TDI 是 TCSEC 在数据库管理系统方面的扩充和解释，并从安全策略、责任、保护和文档 4 个方面进一步描述了每级的安全标准。按照 TCSEC 标准，D 类产品是基本没有安全保护措施的产品，C 类产品只提供了安全保护措施，一般不称为安全产品。B 类以上产品是实行强制存取控制的产品，也是真正意义上的安全产品。所谓安全产品均是指安全级别在 B1 以上的产品，而安全数据库研究原型一般是指安全级别在 B1 级以上的以科研为目的，尚未产品化的数据库管理系统原型。

12.4.2　数据库的完整性设计

数据库完整性是指数据库中数据的正确性和相容性。数据库完整性由各种各样的完整性约束来保证，因此可以说数据库完整性设计就是数据库完整性约束的设计。数据库完整性约束可以通过 DBMS 或应用程序来实现，基于 DBMS 的完整性约束作为模式的一部分存入数据库中。通过 DBMS 实现的数据库完整性按照数据库设计步骤进行设计，而由应用软件实现的数据库完整性则纳入应用软件设计。

1．数据库完整性设计原则

在实施数据库完整性设计时，需要把握以下基本原则。

（1）根据数据库完整性约束的类型确定其实现的系统层次和方式，并提前考虑对系统性能的影响。一般情况下，静态约束应尽量包含在数据库模式中，而动态约束由应用程序实现。

（2）实体完整性约束、参照完整性约束是关系数据库最重要的完整性约束，在不影响系统关键性能的前提下需尽量应用。用一定的时间和空间来换取系统的易用性是值得的。

（3）要慎用目前主流 DBMS 都支持的触发器功能，一方面由于触发器的性能开销较大；另一方面，触发器的多级触发不好控制，容易发生错误，非用不可时，最好使用 Before 型语句级触发器。

（4）在需求分析阶段就必须制定完整性约束的命名规范，尽量使用有意义的英文单词、缩写词、表名、列名及下划线等组合，使其易于识别和记忆，如 CKC_EMP_REAL_INCOME_EMPLOYEE、PK_EMPLOYEE、CKT_EMPLOYEE。如果使用 CASE 工具，一般有默认的规则，可在此基础上修改使用。

（5）要根据业务规则对数据库完整性进行细致的测试，以尽早排除隐含的完整性约束间的冲突和对性能的影响。

（6）要有专职的数据库设计小组，自始至终负责数据库的分析、设计、测试、实施及早期维护。数据库设计人员不仅负责基于 DBMS 的数据库完整性约束的设计实现，还要负责对应用软件实现的数据库完整性约束进行审核。

（7）应采用合适的 CASE 工具来降低数据库设计各阶段的工作量。好的 CASE 工具能够支持整个数据库的生命周期，这将使数据库设计人员的工作效率得到很大提高，同时也容易与用户沟通。

2．数据库完整性的作用

数据库完整性对于数据库应用系统非常关键，其作用主要体现在以下几个方面。

（1）数据库完整性约束能够防止合法用户使用数据库时向数据库中添加不合语义的数据。

（2）利用基于 DBMS 的完整性控制机制来实现业务规则，易于定义，容易理解，而且可以降低应用程序的复杂性，提高应用程序的运行效率。同时，基于 DBMS 的完整性控制机制是集中管理的，因此比应用程序更容易实现数据库的完整性。

（3）合理的数据库完整性设计，能够同时兼顾数据库的完整性和系统的效能。例如装载大量数据时，只要在装载之前临时使基于 DBMS 的数据库完整性约束失效，此后再使其生效，就能保证既不影响数据装载的效率又能保证数据库的完整性。

（4）在应用软件的功能测试中，完善的数据库完整性有助于尽早发现应用软件的错误。

（5）数据库完整性约束可分为 6 类：列级静态约束、元组级静态约束、关系级静态约束、列级动态约束、元组级动态约束和关系级动态约束。动态约束通常由应用软件来实现。不同 DBMS 支持的数据库完整性基本相同，Oracle 支持的基于 DBMS 的完整性约束如表 12-1 所示。

表 12-1　Oracle 支持的基于 DBMS 的完整性约束

Oracle 支持的完整性约束	对应的完整性约束类型	备　注
非空约束（Not Null）	列级静态约束	
唯一码约束（Unique Key）	列级静态约束 元组级静态约束	通过唯一性索引来实现
主键约束（Primary Key）	关系级静态约束	
参照完整性约束（Referential）	关系级静态约束	可定义 5 种不同的动作，Restrict、Wet to Null、Set to Default、Cascade、No Action
检查约束（Check）	列级静态约束 元组级静态约束	可定义在列表或表上
通过触发器来实现的约束	全部 6 类完整性约束	关系级动态约束可以通过调用包含事务的存储过程来实现。如果出现性能问题，需要改由应用软件来实现

3．数据库完整性设计示例

一个好的数据库完整性设计，首先需要在需求分析阶段确定要通过数据库完整性约束实现的业务规则。然后在充分了解特定 DBMS 提供的完整性控制机制的基础上，依据整个系统的体系结构和性能要求，遵照数据库设计方法和应用软件设计方法，合理选择每个业务规则的实现方式。最后，认真测试，排除隐含的约束冲突和性能问题。基于 DBMS 的数据库完整性设计大体分为以下几个阶段。

1）需求分析阶段

经过系统分析员、数据库分析员和用户的共同努力，确定系统模型中应该包含的对象，如人事及工资管理系统中的部门、员工和经理等，以及各种业务规则。

在完成寻找业务规则的工作之后，确定要作为数据库完整性的业务规则，并对业务规则进行分类。其中作为数据库模式一部分的完整性设计按下面的过程进行，而由应用软件来实现的数据库完整性设计将按照软件工程的方法进行。

2）概念结构设计阶段

概念结构设计阶段是将依据需求分析的结果转换成一个独立于具体 DBMS 的概念模型，即实体关系图（Entity-Relationship Diagram，ERD）。在概念结构设计阶段就要开始数据库完整性设计的实质阶段，因为此阶段的实体关系将在逻辑结构设计阶段转化为实体完整性约束和参照完整性约束，到逻辑结构设计阶段将完成设计的主要工作。

3）逻辑结构设计阶段

此阶段就是将概念结构转换为某个 DBMS 所支持的数据模型，并对其进行优化，包括对关系模型的规范化。此时，依据 DBMS 提供的完整性约束机制，对尚未加入逻辑结构中的完整性约束列表，逐条选择合适的方式加以实现。

在逻辑结构设计阶段结束时，作为数据库模式一部分的完整性设计也就基本完成了。每种业务规则都可能有好几种实现方式，应该选择对数据库性能影响最小的一种，有时需通过实际测试来决定。

12.5 案例：电子商务系统的安全性设计

本节以一个具体的电子商务系统——高性能的 RADIUS，来阐明电子商务系统的安全设计的基本原理和设计方法。

1．原理介绍

AAA（Authentication Authorization and Accounting，验证、授权和审记）是运行于宽带网络接入服务器上的客户端程序。AAA 提供了一个用来对验证、授权和审记三种安全功能进行配置的一致的框架，实际上是对网络安全的一种管理。这里的网络安全主要指访问控制，包括哪些用户可以访问网络服务器？如何对正在使用网络资源的用户进行记账？下面简单介绍验证、授权和记账的作用。

（1）验证（Authentication）：验证用户是否可以获得访问权，认证信息包括用户名、用户密码和认证结果等。

（2）授权（Authorization）：授权用户可以使用哪些服务，授权包括服务类型及服务相关信息等。

（3）审记（Accounting）：记录用户使用网络资源的情况，用户 IP 地址、MAC 地址掩码等。

RADIUS 服务器负责接收用户的连接请求，完成验证并把用户所需的配置信息返回给 BAS 建立连接，从而可以获得访问其他网络的权限时，BAS 就起到了认证用户的作用。BAS 负责把用户之间的验证信息传递通过密钥的参与来完成。用户的密码加密以后才能在网上传递，以避免用户的密码在不安全的网络上被窃取。

例如，用户 A 请求得到某些服务（如 PPP、 Telnet 和 Rlogin 等），但必须通过 BAS，由 BAS 依据某种顺序与所连接服务器通信从而进行验证。用户 A 通过拨号进入 BAS，然后 BAS 按配置好的验证方式（如 PPP、 PAP 和 CHAP 等）要求用户 A 输入用户名和密码等信息。用户 A 终端出现提示，用户按提示输入。通过与 BAS 的连接，BAS 得到这些信息。而后 BAS 把这些信息传递给响应验证或记账的服务器，并根据服务器的响应来决定用户是否可以获得他所请求的服务。

一个网络允许外部用户通过宽带网对其进行访问，这样用户在地理上可以分散。大

量分散用户可以通过 DSL Modem 等从不同的地方对这个网络进行随机的访问，用户可以把自己的信息传递给这个网络，也可以从这个网络得到自己想要的信息。由于存在内外的双向数据流动，网络安全就成为很重要的问题，因此对信息进行有效管理是必要的。管理的内容包括用户是否可以获得访问权、用户可以允许使用哪些服务，以及如何对使用网络资源的用户进行计费。AAA 很好地完成了这 3 项任务。

2．软件架构设计

RADIUS 软件主要应用于宽带业务运营的支撑管理，是一个需要可靠运行且高安全级别的软件支撑系统。RADIUS 软件的设计还需要考虑一个重要的问题，即系统高性能与可扩展性。

电信数据业务的开展随着我国宽带业务的开展，在宽带接入方式、宽带业务管理等诸多方面均会发生变化，以适应市场的发展。业务的发展对 RADIUS 软件架构的设计就是重中之重了，其设计将会直接影响系统可持续建设的质量与成本。通过深入分析，高性能的 RADIUS 软件架构核心如图 12-12 所示。

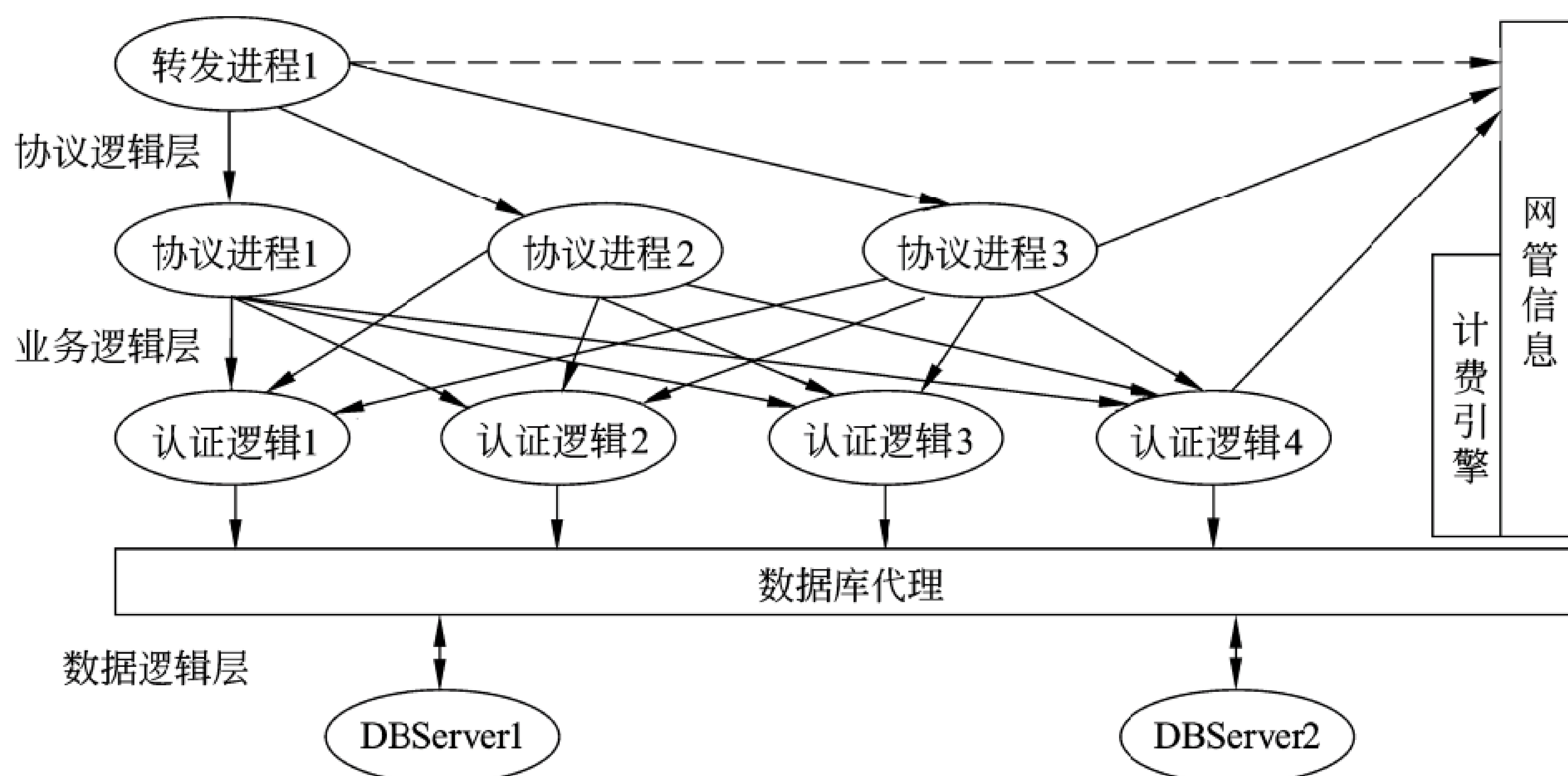


图 12-12　RADIUS 软件架构核心逻辑性

RADIUS 软件架构分为三个层面：协议逻辑层、业务逻辑层和数据逻辑层。

协议逻辑层主要实现 RFC 框架中的内容，处理网络通信协议的建立、通信和停止方面的工作。在软件功能上，这个部分主要相当于一个转发引擎，起到分发处理的内容分发到不同的协议处理过程中，这一层的功能起到了协议与业务处理的分层处理的作用。

业务逻辑层的设计是 RADIUS 软件架构设计的核心部分，架构设计的好坏将直接关系到应用过程中能否适应 RADIUS 协议扩展部分的实现，更重要的是会直接影响到用户单位的业务能否顺利开展。协议处理进程主要是对转发引擎发来的包进行初步分析，并根据包的内容进一步分发到不同的业务逻辑处理进程。协议处理进程可以根据项目的情

况，配置不同的协议进程数，提高包转发与处理的速度。业务逻辑进程分为认证、计费和授权三种类型，不同的业务逻辑进程可以接收不同协议进程之间的信息并进行处理。转发进程与协议进程之间采用共享内存的方法，实现进程之间的通信。协议进程与业务逻辑处理进程之间采用进程加线程的实现方法，这样实现的好处在于不需要对业务处理线程进行应用软件层面的管理，而由 UNIX 系统进行管理，进一步提高应用系统处理的效率与质量。

数据逻辑层需要对来自业务逻辑处理线程统一管理与处理数据库代理池的数据，由数据库代理池统一连接数据库，以减少对数据库系统的压力。同时减小了系统对数据库的依赖性，增强了系统适应数据库系统的能力。

RADIUS 软件分层架构的实现，一是对软件风险进行了深入的分析，并且在软件实现的过程中得到更多的体现；二是可以构建一个或多个重用的构件单元，同时也可以继承原来的成果。BAS 和 RADIUS 之间验证信息的传递是通过密钥的参与来完成的。从原来的窄带拨号上网到现在的宽带接入、无线接入，在信息加密方面从传统的 MD5、PAP 和 CHAP 方式增加了 EAP-tls、P-ttls 和 EAP-sim 等多种格式。基于分层架构的协议处理进程有自然的灵活性，可快速适应 RFC 指南中增加的内容。

RADIUS 的功能，一是实际处理大量用户并发的能力，二是软件架构的可扩展性。负载均衡是提高 RADIUS 软件性能的有效方法，它主要完成以下任务。

（1）解决网络拥塞问题，就近提供服务，实现地理位置无关性。

（2）为用户提供更好的访问质量。

（3）提高服务器响应速度。

（4）提高服务器及其他资源的利用效率。

（5）避免了网络关键部位出现单点失效。

当同时在线的宽带用户量巨大时，BAS 发送给后台 RADIUS 的用户数据更新包的数量会急剧增加，RADIUS 服务器的处理能力就成为性能瓶颈。当包的数量大于 RADIUS 服务器的处理能力时，就会出现丢包，造成用户数据的丢失或不完整。

通过代理转发的方式，把从 BAS 发送过来的数据包平均转发到其他 RADIUS 服务器中进行处理，实现 RADIUS 服务器之间的负载均衡。

RADIUS 高性能还体现在自我管理的功能，该功能包括 UNIX 守护管理监控和进程管理监控。在有故障时，服务进程能内部调度进程，以协调进程的工作情况。同时对 RADIUS 报文进行 SNMP 的代理管理，向综合网络管理平台实时发送信息。

第 13 章　系统的可靠性设计

随着软件复杂度的增加，软件设计的正确性验证成本也越来越高。可靠和可信的计算模型首先在军事和高要求的商业系统中开始研究，可靠性和其他质量属性一样是衡量软件架构的重要指标。实践证明，保障软件可靠性最有效、最经济、最重要的手段是在软件设计阶段采取措施进行可靠性控制。本章探讨软件可靠性的概念、建模与管理方法。

13.1　软件可靠性

13.1.1　软件可靠性概述

在现代军事和商用系统中，以软件为核心的产品得到了广泛的应用。随着系统中软件成分的不断增加，使得系统对软件的依赖性越来越强，对软件可靠性的要求也越来越高。目前，硬件可靠性测试技术和评估手段日趋成熟，硬件可靠性评估模型经过长期的实践积累，已经得到了业界的认可。但是，由于软件和硬件存在着巨大的差异性，硬件的可靠性测试和评估技术，并不能完全应用于对软件的可靠性的测试和评估中。因此，软件可靠性技术研究成为当今可靠性工程研究领域中一个重要领域。

国外从 20 世纪 60 年代后期开始加强对软件可靠性的研究工作，经过 40 多年的研究，推出了各种可靠性模型和预测方法，于 1990 年前后形成了较为系统的软件可靠性工程体系。同时，从 20 世纪 80 年代中期开始，西方各主要工业强国均确立了专门的研究计划和课题，如英国的 AIVEY（软件可靠性和度量标准）计划、欧洲的 ESPRIT（欧洲信息技术研究与发展战略）计划、SPMMS（软件生产和维护管理保障）课题和 Eureka（尤里卡）计划等。每年，都有大量的人力物力投入到软件可靠性研究项目中，并取得了一定的成果。

国内对于软件可靠性的研究工作起步较晚，在软件可靠性量化理论、度量标准（指标体系）、建模技术、设计方法和测试技术等方面与国外差距较大。

目前，软件可靠性管理方面还没有建立起具有权威性的管理体系和规范。例如，如何描述软件可靠性，如何测试、评估、设计和提高等。由于目前国内外对于软件可靠性模型的研究多集中在软件的开发阶段及测试与评估阶段的可靠性模型，而且现有的模型也多来源于硬件可靠性评估，与软件可靠性评估存在较大的差距，所以从事软件可靠性测试与评估研究是一个有理论价值和实际意义的工作。总的来说，软件可靠性工程研究虽然得到了普遍的重视，但仍然不是很成熟，还处于发展阶段。

13.1.2 软件可靠性的定义

可靠性（Reliability）是指产品在规定的条件下和规定的时间内完成规定功能的能力。

按照产品可靠性的形成，可靠性可分为固有可靠性和使用可靠性。固有可靠性是通过设计、制造赋予产品的可靠性；使用可靠性既受设计、制造的影响，又受使用条件的影响。一般使用可靠性总低于固有可靠性。

软件与硬件有很多不同点，但从可靠性的角度来看，它们主要有如下 4 个不同点。

（1）复杂性。软件内部逻辑高度复杂，硬件则相对简单，这就在很大程度上决定了设计错误是导致软件失效的主要原因，而导致硬件失效的可能性则很小。

（2）物理退化。软件不存在物理退化现象，硬件失效则主要是由于物理退化所致。这就决定了软件正确性与软件可靠性密切相关，一个正确的软件任何时刻均可靠。然而，一个正确的硬件元器件或系统，则可能在某个时刻失效。

（3）唯一性。软件是唯一的，软件复制不改变软件本身，而任何两个硬件不可能绝对相同。这就是为什么概率方法在硬件可靠性领域取得巨大成功，而在软件可靠性领域不令人满意的原因。

（4）版本更新较快。硬件的更新周期通常较慢，硬件产品一旦定型一般就不会更改，而软件产品通常受需求变更、软件缺陷修复的需要，造成软件版本更新较快，这也给软件可靠性评估带来较大的难度。

尽管这样，软件仍然是一种具有特殊属性的产品，因此，也可以按照上面的产品可靠性定义来框架性地描述软件的可靠性。

1983 年，美国 IEEE 计算机学会对“软件可靠性”做出了更为明确的定义，随后，此定义经美国标准化研究所批准为美国的国家标准。在 1989 年，我国国家标准 GB/T-11457 也采用了这个定义。这个定义就是：在规定的条件下，在规定的时间内，软件不引起系统失效的概率，该概率是系统输入和系统使用的函数，也是软件中存在的缺陷函数；系统输入将确定是否会遇到已存在的缺陷（如果缺陷存在的话）。

简言之，就是在规定的时间周期内，在所述条件下程序执行所要求的功能的能力。显而易见，美国 IEEE 计算机学会关于“软件可靠性”的定义仍然沿用了“产品可靠性”的定义，但有了更具体的定位和更深入的描述。

下面来分析一下软件可靠性的框架性定义。

（1）规定的时间。

软件可靠性只是体现在其运行阶段，所以将“运行时间”作为“规定的时间”的度量。“运行时间”包括软件系统运行后工作与挂起（开启但空闲）的累计时间。由于软件运行的环境与程序路径选取的随机性，软件的失效为随机事件，所以运行时间属于随机变量。

（2）规定的条件。

规定的条件主要指软件的运行环境。它涉及软件系统运行时所需的各种支持要素，如支持硬件平台（服务器、台式机和网络平台等）、操作系统、数据库管理系统、中间件，以及其他支持软件、输入数据格式和范围及操作规程等。不同的环境条件下软件的可靠性是不同的，具体地说，规定的环境条件主要是描述软件系统运行时计算机的配置情况及对输入数据的要求，并假定其他一切因素都是理想的。有了明确规定的环境条件，还可以有效地判断软件失效的责任在用户方还是开发方。

（3）所要求的功能。

软件可靠性还与规定的任务和功能有关。由于要完成的任务不同，软件的运行情况会有所区别，则调用的子模块就不同（包括程序选择路径不同），其可靠性也就可能不同。所以，要准确度量软件系统的可靠性，必须先明确它的任务和功能。

（4）“软件可靠性”定义具有以下特点。

① 用内在的“缺陷”和外在的“失效”关系来描述可靠性，更能深刻地体现软件的本质特点。

② 定义使人们对软件可靠性进行量化评估成为可能。对于软件的可靠性这样一个质量特性，很难用一个明确直观的数值去体现。而依据这个定义，我们有可能通过分析影响可靠性的因素，用函数的形式，按照不同的目的建立各种数学模型去分析软件可靠性。

③ 用概率的方法去描述可靠性是比较科学的。前面讲到，软件失效是随机的外部表现，完全是一个随机事件，而软件缺陷是软件固有的没有损耗的内在特点。定义用规定时间内其操作不出现软件失效的概率，也就是输入未碰到软件缺陷的概率来描述可靠性，这种方法就是用概率来描述纯粹的随机事件，是比较合理的，也是可行的。

13.1.3　软件可靠性的定量描述

从软件可靠性的定义可以看到，软件的可靠性可以基于使用条件、规定时间、系统输入、系统使用和软件缺陷等变量构建的数学表达式。下面从可靠性定义中的术语“规定时间”、“失效概率”开始，探讨软件可靠性的定量描述，并相应地引入一些概念。

1．规定时间

对于“规定时间”有三种概念：一种是自然时间，也就是日历时间，指我们日常计时用的年、月、周、日等自然流逝的时间段；一种是运行时间，指软件从启动开始，到运行结束的时间段；最后一种是执行时间，指软件运行过程中，中央处理器（CPU）执行程序指令所用的时间总和。

例如，某单位有一套供会计人员使用的财务软件，我们来关注一整天的时间，上午 9:00 上班开机运行，下午 5:00 下班退出程序。在这里，自然时间是一天，也就是 24 小时，运行时间是 8 个小时，而 CPU 处理程序的执行时间可能不到 2 小时，这要视会计的

业务繁忙状况、使用软件的频度和软件本身的设计而定。

很明显，在这三种时间中，我们使用执行时间来度量软件的可靠性最为准确，效果也最好。如果运行的软件系统处于一种相对稳定的工作状态，可以根据一定的经验值，按一定的换算比例，对这三种时间进行折算。

2．失效概率

我们把软件从运行开始，到某一时刻 t 为止，出现失效的概率看作关于软件运行时间的一个随机函数，用 $F(t)$表示。根据我们对软件可靠性的分析，函数 $F(t)$有如下特征。

（1）$F(0)=0$，即软件运行初始时刻失效概率为 0。

（2）$F(t)$在时间域$(0,+\infty)$上是单调递增的。

（3）$F(+\infty)=1$，即失效概率在运行时间不断增长时趋向于 1，这也和“任何软件都存在缺陷”的思想相吻合。

为了简化分析，把 $F(t)$看作关于时间 t 的一个连续函数，并且可导。

3．可靠度

我们用来表示可靠性最为直接的方式就是可靠度，根据可靠性的定义，可靠度就是软件系统在规定的条件下、规定的时间内不发生失效的概率。如果用 $F(t)$来表示到 t 时刻止，软件不出现失效的概率，则可靠度的公式为

$$R(t)=1-F(t) \tag{13-1}$$

同样，我们知道 $R(0)=1$，$R(+\infty)=0$。

4．失效强度

失效强度（Failure Intensity）的物理解释就是单位时间软件系统出现失效的概率。在 t 时刻到 $t+\Delta t$ 时刻之间软件系统出现失效的平均概率为$(F(t+\Delta t)-F(t))/\ \Delta t$，当 Δt 趋于很小时，就表现为 t 时刻的失效强度。用 $f(t)$表示失效强度函数，则

$$f(t)=\lim_{\Delta t\to 0}\frac{F(t+\Delta t)}{\Delta t}=F'(t) \tag{13-2}$$

5．失效率

失效率（Failure Rate）又称风险函数（Hazard Function），也可以称为条件失效强度，物理解释就是在运行至此软件系统未出现失效的情况下，单位时间软件系统出现失效的概率。具体用数学用语来描述，就是当软件在 0～t 时刻内没有发生失效的条件下，t 时刻软件系统的失效强度。用 $\lambda(t)$表示失效率，则

$$f(t)=\lambda(t)\cdot R(t) \tag{13-3}$$

代入公式（13-1）可得从可靠度到失效率的转换表达式

$$\lambda(t)=\frac{f(t)}{R(t)}=\frac{F'(t)}{R(t)}=\frac{R'(t)}{R(t)} \tag{13-4}$$

6．可靠度与失效率之间的换算

我们知道，在 0 时刻，可靠度 $R(0)$为 1，对公式（13-4）一阶常微分方程求解可得

$$R(t)=e^{-\int_0^1 \lambda(s)\mathrm{d}s} \tag{13-5}$$

假设软件系统的失效率为常数时，由公式 13-5 可得

$$R(t)=e^{-\lambda t} \tag{13-6}$$

当失效率 $\lambda(t)$与时间 t 之积，也就是 $t\lambda(t)<0.05$ 时，公式（13-6）可简化为

$$R(t)\approx 1-\lambda(t)\bullet t \tag{13-7}$$

这样计算，误差在 2.5%之内。

由公式（13-6）可得，从可靠度到失效强度的转换公式

$$\lambda(t)=\frac{\ln[R(t)]}{t} \tag{13-8}$$

当可靠度 $R(t)>0.95$ 时，公式（13-6）可简化为

$$\lambda(t)=\frac{1-R(t)}{t} \tag{13-9}$$

这样计算，误差在 2.5%之内。

7．平均无失效时间

平均无失效时间（Mean Time to Failure，MTTF）就是软件运行后，到下一次出现失效的平均时间。通常平均无失效时间更能直观地表明一个软件的可靠程度。用 θ 表示平均无失效时间 MTTF，则可得

$$\theta=\int_0^{+\infty} R(t)\mathrm{d}t \tag{13-10}$$

代入关于失效率的换算公式，可得

$$\theta=\int_0^{+\infty} e^{-\lambda(t)\bullet t}\mathrm{d}t \tag{13-11}$$

当失效率为一个常数时，可得

$$\theta=\frac{1}{\lambda} \tag{13-12}$$

当讨论完对软件可靠性的定量描述问题之后，需要对软件可靠度这个直接反映软件可靠性的度量指标作下列补充说明。

（1）描述的软件对象必须明确，即需指明它与其他软件的界限。

（2）软件失效必须明确定义。

（3）必须假设硬件无故障（失效）和软件有关变量的输入值正确。

（4）运行环境包括硬件环境、软件支持环境和确定的软件输入域。

（5）规定的时间必须指明时间基准，可以是自然时间（日历时间）、运行时间、执行时间（CPU 时间）或其他时间基准。

（6）软件无失效运行的机会通常以概率度量，但也可以模糊数学中的可能性加以度量。

（7）上述定义是在时间域上进行的，这时软件可靠度是一种动态度量。也可以是在

数据域上将软件可靠度定义为一种表态度量，表示软件成功执行一个回合的概率。软件回合（Run）是指软件在规定环境下的一个基本执行过程，如给定一组输入数据，到软件给定相应的输出数据这一过程。软件回合是软件运行最小的、不可分的执行单位，软件的运行过程由一系列软件回合组成。

（8）有时将软件运行环境简单地理解为软件运行剖面（Operational Profile）。欧空局（ESA）标准 PSS-01-21（1991）“ESA 软件产品保证要求”中，定义“软件运行剖面”为：“对系统使用条件的定义。系统的输入值都用其按时间的分布或按它们在可能输入范围内的出现概率的分布来定义”。简单来说，运行剖面定义了关于软件可靠性描述中的“规定条件”，也就是相当于可靠性测试中需要考虑的测试环境、测试数据等一系列问题。

13.1.4 可靠性目标

前面定量分析软件的可靠性时，使用失效强度来表示软件缺陷对软件运行的影响程度。然而在实际情况中，对软件运行的影响程度不仅取决于软件失效发生的概率，还和软件失效的严重程度有很大关系。这里引出另外一个概念——失效严重程度类（Failure Severity Class）。

失效严重程度类就是对用户具有相同程度影响的失效集合。

对失效严重程度的分级可以按照不同的标准进行，最为常见的是按对成本影响、对系统能力的影响等标准划分软件失效的严重程度类。

对成本的影响可能包括失效引起的额外运行成本、修复和恢复成本、现有或潜在的业务机会的损失等。由于失效严重程度类的影响分布很广泛，为了按照一定数量的等级去定义失效严重程度类，通常用数量级去划分等级。

表 13-1 给出了一个按照对成本的影响划分失效严重程度类的例子，这个例子涉及到的软件系统是某电子商务运营系统。

表 13-1 按照对成本的影响划分失效严重程度类

失效严重程度	定义（人民币万元）	失效严重程度类	定义（人民币万元）
1	成本＞100	4	0.1＜成本≤1
2	10＜成本≤100	5	成本＜0.1
3	1＜成本≤10		

对系统能力的影响常常表现为关键数据的损失、系统异常退出、系统崩溃、导致用户操作无效等。对于不同性质的软件系统，相同的表现可能造成的失效严重程度是不同的，例如对可用性要求较高的系统，导致长时间停机的失效常常会划分到较高的严重级别中去。

表 13-2 给出了一个按照对系统能力的影响划分失效严重程度类的例子，这个例子涉及到的软件系统是某电信实时计费系统。

表 13-2　按照对系统能力的影响划分失效严重程度类

失效严重程度类	定　　义
1	系统崩溃，重要数据不可恢复
2	系统出错停止响应，重要数据可恢复
3	用户重要操作无响应，可恢复
4	部分操作无响应，但可用其他操作方式替代

有了失效严重程度的划分，现在可以结合失效强度来定量地表示一个软件系统的可靠性目标了。

可靠性目标是指客户对软件性能满意程度的期望。通常用可靠度、故障强度和平均失效时间（MTTF）等指标来描述，根据不同项目的不同需要而定。建立定量的可靠性指标需要对可靠性、交付时间和成本进行平衡。为了定义系统的可靠性指标，必须确定系统的运行模式，定义故障的严重性等级，确定故障强度目标。

例如，对于表 13-2 的例子，可以根据经验和用户的需求确定软件系统需要达到的可靠程度，按照前面的公式，换算成失效强度和平均无失效时间，如表 13-3 所示。

表 13-3　可靠性目标参考表

失效严重程度类	可靠性要求/%	失 效 强 度	平均无失效时间
1	99.9999	10^{-6}	114 年
2	99.99	10^{-4}	417 天
3	99	10^{-1}	4 天
4	90	1	9 小时

13.1.5　可靠性测试的意义

软件可靠性问题已被越来越多的软件工程专家所重视，人们已开始投入大量的人力、物力去研究软件可靠性的设计、评估和测试等课题。以下多个方面可以反映出软件可靠性问题对软件工程实践、乃至对生产活动和社会活动产生的深远影响。

（1）软件失效可能造成灾难性的后果。一个最显著的例子就是由于控制系统的 Fortran 程序中少了个逗点，致使控制系统未能发出正确的指令，最终使美国的一次宇宙飞行失败。而目前由于计算机和软件在各行各业中应用的日益广泛和深入，例如军用作战系统、民航指挥系统、银行支付系统和交通调控系统等，一旦发生严重级别的软件失效，轻则造成经济损失，重则危及人们的生命安全，危害国家安全。

（2）软件的失效在整个计算机系统失效中的比例较高。某研究机构曾经作过统计，在计算机系统的失效中，有 80%和软件有关。原因是软件系统的内容结构太复杂了，一个较简单的程序，其所有的路径数就可能是一个天文数字。在软件开发的过程中，很难用全路径覆盖方式的测试去发现软件系统中隐藏的所有缺陷，也就是说，很难完全排除软件缺陷。

（3）相比硬件可靠性技术，软件可靠性技术很不成熟，这就加剧了软件可靠性问题的重要性。例如在硬件可靠性领域，故障树分析（Fault Tree Analysis，FTA）、失效模式

与效应分析（Failure Made And Effect Analysis，FMEA）技术等比较成熟，容错技术也有广泛应用，但在软件可靠性领域，这些技术似乎尚未定型。

（4）与硬件元器件成本急剧下降形成鲜明对比的是，软件费用呈有增无减的势头，而软件可靠性问题是造成费用增长的主要原因之一。

（5）计算机技术获得日益广泛的应用，随着计算机应用系统中软件成分的不断增加，使得系统对于软件的依赖性越来越强，软件对生产活动和社会生活的影响越来越大，从而增加了软件可靠性问题在软件工程领域乃至整个计算机工程领域的重要性。

软件可靠性问题的重要性凸显了发展以发现软件可靠性缺陷为目的的可靠性设计与测试技术的迫切性。

13.1.6 广义的可靠性测试与狭义的可靠性测试

广义的软件可靠性测试是指为了最终评价软件系统的可靠性而运用建模、统计、试验、分析和评价等一系列手段对软件系统实施的一种测试。一个完整的软件可靠性测试包括图 13-1 所示的过程。

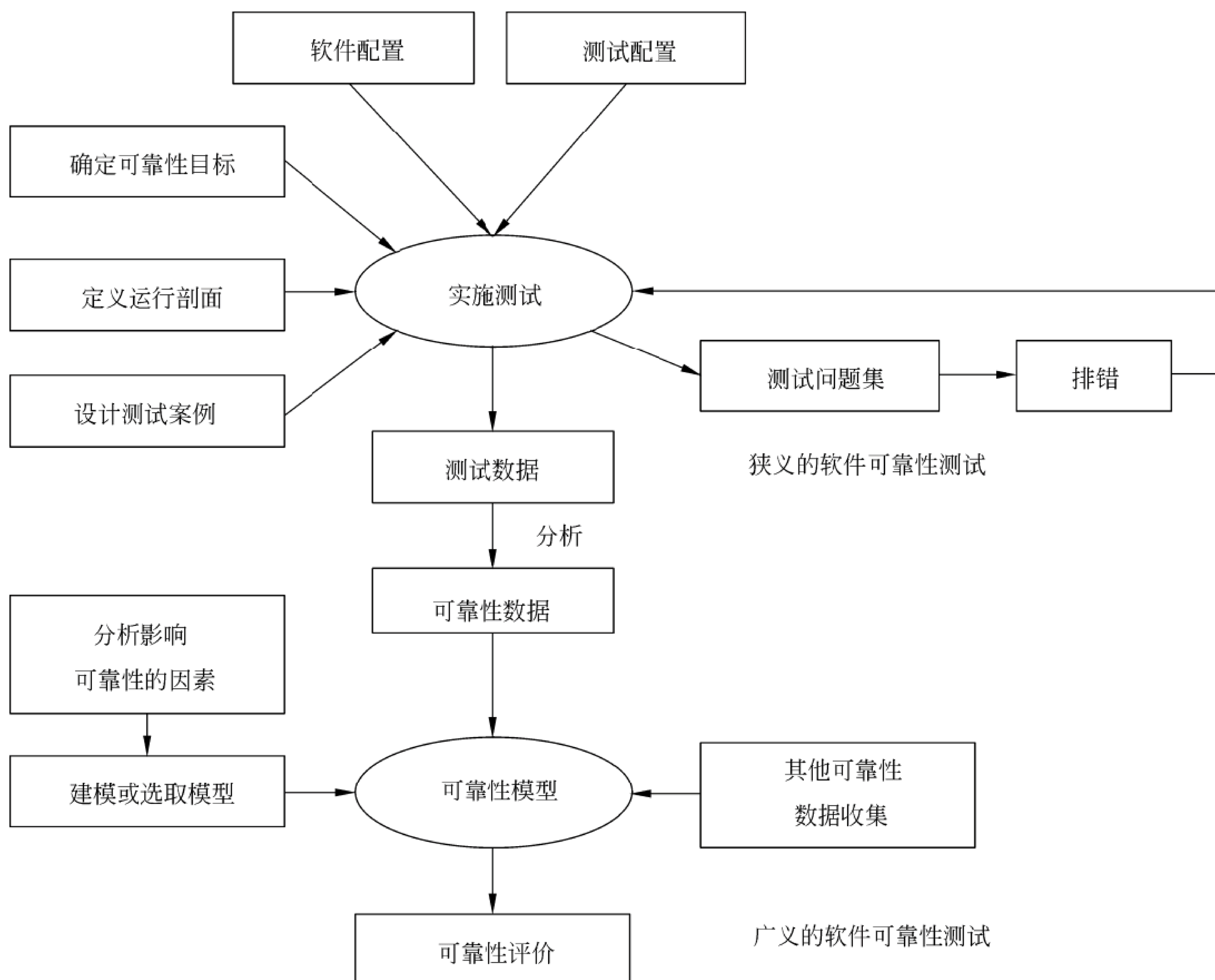


图 13-1 广义的软件可靠性测试

狭义的软件可靠性测试是指为了获取可靠性数据，按预先确定的测试用例，在软件的预期使用环境中，对软件实施的一种测试。狭义的软件可靠性测试也叫"软件可靠性试验（Software Reliability Test）"，它是面向缺陷的测试，以用户将要使用的方式来测试软件，每一次测试代表用户将要完成的一组操作，使测试成为最终产品使用的预演。这就使得所获得的测试数据与软件的实际运行数据比较接近，可用于软件可靠性评价。

其实，软件可靠性测试是软件测试的一种形式，和易用性测试、性能测试、标准符合性测试等前面介绍的测试类型一样，是针对软件的某个重要质量特性，使用一定的测试用例对软件进行测试的过程。

可靠性测试是对软件产品的可靠性进行调查、分析和评价的一种手段。它不仅仅是为了用测试数据确定软件产品是否达到可靠性目标，还要对检测出的失效的分布、原因及后果进行分析，并给出纠正建议。总的来说，可靠性测试的目的可归纳为以下三个方面。

（1）发现软件系统在需求、设计、编码、测试和实施等方面的各种缺陷。

（2）为软件的使用和维护提供可靠性数据。

（3）确认软件是否达到可靠性的定量要求。

13.2　软件可靠性建模

13.2.1　影响软件可靠性的因素

在讲到软件可靠性评估的时候，我们不得不提到软件可靠性模型。软件可靠性模型（Software Reliability Model）是指为预计或估算软件的可靠性所建立的可靠性框图和数学模型。建立可靠性模型是为了将复杂系统的可靠性逐级分解为简单系统的可靠性，以便于定量预计、分配、估算和评价复杂系统的可靠性。

为了构建软件的可靠性模型，首先要来分析一下影响软件可靠性的因素。影响软件可靠性的因素是纷杂而众多的，甚至包括技术以外的许多因素。首先必须考虑影响软件可靠性的主要因素：缺陷的引入、发现和清除。缺陷的引入主要取决于软件产品的特性和软件的开发过程特性。软件产品的特性指软件本身的性质，开发过程特性包括开发技术、开发工具、开发人员的水平、需求的变化频度等。缺陷的发现依靠用户对软件的操作方式、运行环境等，也就是运行剖面。缺陷的清除依赖于失效的发现和修复活动及可靠性方面的投入。

从技术的角度来看，影响软件可靠性的主要因素如下。

（1）运行剖面（环境）。软件可靠性的定义是相对运行环境而言的，一样的软件在不同的运行剖面下，其可靠性的表现是不一样的。

（2）软件规模。也就是软件的大小，一个只有数十行代码的软件和几千万行代码的

软件是不能相提并论的。

（3）软件内部结构。结构对软件可靠性的影响主要取决于软件结构的复杂程度，一般来说，内部结构越复杂的软件，所包含的软件缺陷数就可能越多。

（4）软件的开发方法和开发环境。软件工程表明，软件的开发方法对软件的可靠性有显著影响。例如，与非结构方法相比，结构化方法可以明显减少软件的缺陷数。

（5）软件的可靠性投入。软件在生命周期中可靠性的投入包括开发者在可靠性设计、可靠性管理、可靠性测试和可靠性评价等方面投入的人力、资金、资源和时间等。经验表明，在早期重视软件可靠性并采取措施开发出来的软件，可靠性有明显的提高。

总之，有许许多多的因素影响着软件的可靠性，有些至今也无法确定它们与软件可靠性之间的定量关系，甚至定性关系也不甚清楚。

13.2.2 软件可靠性建模方法

一个软件可靠性模型通常（但不是绝对）由以下几部分组成。

（1）模型假设。模型是实际情况的简化或规范化，总要包含若干假设，例如测试的选取代表实际运行剖面，不同软件失效独立发生等。

（2）性能度量。软件可靠性模型的输出量就是性能度量，如失效强度、残留缺陷数等。在软件可靠性模型中性能度量通常以数学表达式给出。

（3）参数估计方法。某些可靠性度量的实际值无法直接获得，例如残留缺陷数，这时需通过一定的方法估计参数的值，从而间接确定可靠性度量的值。当然，对于可直接获得实际值的可靠性度量，就无需参数估计了。

（4）数据要求。一个软件可靠性模型要求一定的输入数据，即软件可靠性数据。不同类型的软件可靠性模型可能要求不同类型的软件可靠性数据。

绝大多数的模型包含三个共同假设。这些假设至今主宰着软件可靠性建模的研究发展，人们尚未找到克服这些假设局限性的有效方法。

（1）代表性假设。此假设认为软件测试用例的选取代表软件实际的运行剖面，甚至认为测试用例是独立随机地选取。此假设实质上是指可以用测试产生的软件可靠性数据预测运行阶段的软件可靠性行为。

（2）独立性假设。此假设认为软件失效是独立发生于不同时刻，一个软件失效的发生不影响另一个软件失效的发生。例如在概率范畴，假设相邻软件失效间隔构成一组独立随机变量，或假设一定时间内软件失效次数构成一个独立增量过程。在模糊数学范畴，则相邻软件失效间隔构成一组不相关的模糊变量。

（3）相同性假设。此假设认为所有软件失效的后果（等级）相同，即建模过程只考虑软件失效的具体发生时刻，不区分软件的失效严重等级。

软件可靠性模型要描述失效过程对上一节所分析的因素的一般依赖形式。由于这些因素大多数在本质上是概率性的，并且表现与时间相关联，所以通过失效数据的概率分

布和随机过程随时间的变化的特性来整体区分软件可靠性模型。

我们常常通过下面估计或预测的方法来确定模型的参数。估计是通过收集到的失效数据进行统计分析，利用一定的推导过程归纳出模型的参数；预测则是使用软件产品自身的属性和开发过程来确定模型的参数，这种方法可以在开始执行程序前完成。

确定了模型的参数后，就可以来表示失效过程的很多不同的特性。例如，大多数模型都会对如下的内容进行解析表达。

（1）任何时间点所经历的平均失效数。

（2）一段时间间隔内的平均失效数。

（3）任何时间点的失效强度。

（4）失效区间的概率分布。

在对将来的故障行为进行预测时，应保证模型参数的值不发生变化。如果在进行预测时发现引入了新的错误，或修复行为使新的故障不断发生，就应停止预测，并等足够多的故障出现后，再重新进行模型参数的估计。否则，这样的变化会因为增加问题的复杂程度而使模型的实用性降低。

一般来说，软件可靠性模型是以在固定不变的运行环境中运行的不变的程序作为估测实体的。这也就是说，程序的代码和运行剖面都不发生变化，但它们往往总要发生变化的，于是在这种情况之下，就应采取分段处理的方式来进行工作。因此，模型主要集中注意力于排错。但是，也有的模型具有能处理缓慢地引进错误情况的能力。

对于一个已发行并正在运行的程序，应暂缓安装新的功能和对下一次发行的版本的修复。如果能保持一个不变的运行剖面，则程序的故障密度将显示为一个常数。

一般来说，一个好的软件可靠性模型增加了关于开发项目的交流，并对了解软件开发过程提供了一个共同的工作基础。它也增加了管理的透明度和其他令人感兴趣的东西。即使在特殊的情况之下，通过模型做出的预测并不是很精确的话，上面的这些优点也仍然是明显而有价值的。

要建立一个有用的软件可靠性模型必须有坚实的理论研究工作、有关工具的建造和实际工作经验的积累。通常这些工作要许多人一年的工作量。相反，要应用一个好的软件可靠性模型，则要求以极少的项目资源就可以在实际工作中产生好的效益。

一个好的软件可靠性模型应该具有如下重要特性。

（1）基于可靠的假设。

（2）简单。

（3）计算一些有用的量。

（4）给出未来失效行为的好的映射。

（5）可广泛应用。

13.2.3 软件的可靠性模型分类

一个有效的软件可靠性模型应尽可能地将上面所述的因素在软件可靠性建模时加以考虑，尽可能简明地反映出来。自 1972 年第一个软件可靠性分析模型发表的 30 多年来，见之于文献的软件可靠性统计分析模型将近百种。这些可靠性模型大致可分为如下 10 类。

- 种子法模型。
- 失效率类模型。
- 曲线拟合类模型。
- 可靠性增长模型。
- 程序结构分析模型。
- 输入域分类模型。
- 执行路径分析方法模型。
- 非齐次泊松过程模型。
- 马尔可夫过程模型。
- 贝叶斯分析模型。

下面分别对这些模型进行简单介绍。

1．种子法模型

这类模型利用捕获一再捕获抽样技术估计程序中的错误数，在程序中预先有意“播种”一些设定的错误“种子”，然后根据测试出的原始错误数和发现的诱导错误的比例，来估计程序中残留的错误数。其优点是简便易行，缺点是诱导错误的“种子”与实际的原始错误之间的类比性估量困难。

2．失效率类模型

这类模型用来研究程序的失效率，主要有下列内容。

- Jelinski-Moranda 的 De-eutrophication 模型。
- Jelinski-Moranda 的几何 De-eutrophication 模型。
- Schick-Wolverton 模型。
- 改进的 Schick-Wolverton 模型。
- Moranda 的几何泊松模型。
- Goal 和 Okumoto 不完全排错模型。

3．曲线拟合类模型

这类模型用回归分析的方法研究软件复杂性、程序中的缺陷数、失效率、失效间隔时间，包括参数方法和非参数方法两种。

4．可靠性增长模型

这类模型预测软件在检错过程中的可靠性改进，用增长函数来描述软件的改进过

程。这类模型如下。

- Duane 模型。
- Weibull 模型。
- Wagoner 的 Weibull 改进模型。
- Yamada 和 Osaki 的逻辑增长曲线。
- Gompertz 的增长曲线。

5．程序结构分析模型

程序结构模型是根据程序、子程序及其相互间的调用关系，形成一个可靠性分析网络。网络中的每一结点代表一个子程序或一个模块，网络中的每一有向弧代表模块间的程序执行顺序。假定各结点的可靠性是相互独立的，通过对每一个结点可靠性、结点间转换的可靠性和网络在结点间的转换概率，得出该持续程序的整体可靠性。这类模型如下。

- Littewood 马尔可夫结构模型。
- Cheung 的面向用户的马尔可夫模型。

6．输入域分类模型

这类模型选取软件输入域中的某些样本“点”运行程序，根据这些样本点在“实际”使用环境中的使用概率的测试运行时的成功 / 失效率，推断软件的使用可靠性。这类模型的重点（亦是难点）是输入域的概率分布的确定及对软件运行剖面的正确描述。这类模型如下。

- Nelson 模型。
- Bastani 的基于输入域的随机过程模型。

7．执行路径分析方法模型

这类模型的分析方法与上面的模型相似，先计算程序各逻辑路径的执行概率和程序中错误路径的执行概率，再综合出该软件的使用可靠性。Shooman 分解模型属于此类。

8．非齐次泊松过程模型

非齐次泊松过程模型，即 NHPP，是以软件测试过程中单位时间的失效次数为独立泊松随机变量，来预测在今后软件的某使用时间点的累计失效数。这类模型如下。

- Musa 的指数模型。
- Goel 和 Okumoto 的 NHPP 模型。
- S_型可靠性增长模型。
- 超指数增长模型。
- Pham 改进的 NHPP 模型。

9．马尔可夫过程模型

这类模型如下。

- 完全改错的线性死亡模型。

- 不完全改错的线性死亡模型。
- 完全改错的非静态线性死亡模型。

10．贝叶斯模型

这是利用失效率的试验前分布和当前的测试失效信息，来评估软件的可靠性。这是一类当软件可靠性工程师对软件的开发过程有充分的了解，软件的继承性比较好时具有良好效果的可靠性分析模型。这类模型如下。

- 连续时间的离散型马尔可夫链。
- Shock 模型。

另外，Musa 和 Okumoto 依据模型的不同属性对可靠性模型进行以下分类。

- 时间域：有两种，自然或日历时间与执行（CPU）时间。
- 失效数类：取决于无限时间内发生的失效数是有限的还是无限的。
- 失效数分布：相对于时间系统失效数的统计分布形式，主要的两类是泊松分布型和二项分布型。
- 有限类：对有限失效数的类别适用，用时间表示的失效强度的函数形式。
- 无限类：对无限失效数的类别适用，用经验期望失效数表示的失效强度的函数形式。

13.2.4 软件可靠性模型举例

迄今已有数十种模型是根据上一小节中关于模型的分类方法进行的分类，下面将介绍 Jelinski-Moranda 模型的基本思想及其相关的历史背景。

Jelinski-Moranda 模型（JM 模型）是 Z.Jelinski 和 P.Moranda 于 1972 年提出的软件可靠性数学模型，是最具代表性的早期软件可靠性马尔可夫过程的数学模型。随后的许多工作都是在它的基础上对其中与软件开发实际不相适合的地方进行改进而提出来的，所以，JM 模型是具有广泛影响的模型之一。

1．模型假设

JM 模型的基本假设如下。

（1）软件系统中的初始错误个数为一个未知的常数，用 N_0 表示。

（2）可靠性测试中发现的错误立即被完全排除，并且排除过程不引入新的错误，排除时间忽略不计。因此，每次排错之后，N_0 就要减去 1。

（3）在任何一个失效间隔区间，软件系统的失效率与系统中剩余的错误个数成正比，比例常数用 Φ 表示。

其实，最初 Jehnski 和 Moranda 提出的模型假设只有最后一条，前面两个假设是后人根据使用过程中出现的问题归纳总结而来的。

2．函数表达式

根据假设，每发生 1 次失效，错误数都要减去 1，如果用 t_1，t_2，…，t_i 表示从 0 时

刻开始的每次失效间隔时间，那第 i–1 次失效到第 i 次失效之间的失效率为

$$\lambda(t_i)=\Phi(N_0-i+1) \tag{13-13}$$

根据在可靠性定量描述一节的讨论，知道失效强度函数为

$$f(t_i)=\Phi(N_0-i+1)e^{-\Phi(N_0-i+1)t_i} \tag{13-14}$$

可靠度函数为

$$R(t_i)=e^{-\Phi(N_0-i+1)t_i} \tag{13-15}$$

失效概率分布函数为

$$F(t_i)=1-e^{-\Phi(N_0-i+1)t_i} \tag{13-16}$$

3．参数估计

在可靠度函数表达式中含有两个未知参数 Φ 和 N_0，下面运用统计学中的最大似然法来对参数 Φ 和 N_0 进行估算。

由公式（13-15）可得似然函数

$$L(t_1,t_2,\cdots,t_n)=\prod_{i=1}^{n}\Phi(N_0-i+1)e^{-\Phi(N_0-i+1)t_i} \tag{13-17}$$

对公式（13-17）取对数，得到对数似然函数

$$\begin{aligned}\ln L&=\sum_{i=1}^{n}\ln\left[\Phi(N_0-i+1)e^{-\Phi(N_0-i+1)t_i}\right]\\&=\sum_{i=1}^{n}\ln\left[\Phi(N_0-i+1)\right]-\sum_{i=1}^{n}\left[\Phi(N_0-i+1)t_i\right]\\&=\sum_{i=1}^{n}\ln\left[(N_0-i+1)+n\ln\Phi-\sum_{i=1}^{n}\left[\Phi(N_0-i+1)t_i\right]\right]\end{aligned} \tag{13-18}$$

对公式（13-18）中的 N_0 和 Φ 求偏导，并令结果为 0

$$\frac{\partial \ln L}{\partial N_0}=\sum_{i=1}^{n}\frac{1}{N_0-i+1}-\sum_{i=1}^{n}\Phi t_i=0 \tag{13-19}$$

$$\frac{\partial \ln L}{\partial N_0}=\frac{n}{\Phi}=-\sum_{i=1}^{n}(N_0-i+1)t_i=0 \tag{13-20}$$

公式（13-19）可以写成

$$\begin{aligned}\frac{\partial \ln L}{\partial N_0}&=\sum_{i=1}^{n}\frac{1}{N_0-i+1}\\&=\frac{nT}{N_0T-\sum_{i=1}^{n}(i-1)t_i}\\&=\frac{n}{N_0-\frac{1}{T}\sum_{i=1}^{n}(i-1)t_i}\end{aligned} \tag{13-21}$$

公式（13-21）中不含Φ，因而可以由测试收集的数据，计算出$T=\sum_{i=1}^{n}t_i$和$\sum_{i=1}^{n}(i=1)t_i$的值，将它们代入公式（13-21）中，即可先解出N_0的估计值$\hat{N}_0$

$$\begin{aligned}\hat{N}_0&=\frac{1}{N_0}+\frac{1}{N_0-1}+\cdots+\frac{1}{N_0-(n-1)}\\&=\frac{n}{N_0-\sum_{i=1}^{n}t_i\bullet\sum_{i=1}^{n}(i-1)t_i}\end{aligned}\qquad(13\text{-}22)$$

再来解出另一个参数Φ的估计值，令

$$T=\sum_{i=1}^{n}t_i\qquad(13\text{-}23)$$

则从（13-20）中可解出

$$\Phi=\frac{n}{N_0T-\sum_{i=1}^{n}(i-1)t_i}\qquad(13\text{-}24)$$

代入N_0的估计值，可解出Φ的估计值$\hat{\Phi}$

$$\hat{\Phi}=\frac{n}{\hat{N}_0\sum_{i=1}^{n}t_i-\sum_{i=1}^{n}(i-1)t_i}\qquad(13\text{-}25)$$

需要说明的是，软件可靠性是一门正在发展中的分支学科，许多来源于硬件可靠性的理论在软件可靠性研究中并不适用，有关软件可靠性的模型并不成熟，并且应用范围也非常有限，软件可靠性的定量分析方法和数学模型要在实践中不断加以验证和修正，对于不同类型的软件，模型的假设、表示公式及应用方式也有很大的区别。

13.2.5 软件可靠性测试概述

软件测试者可以使用很多方法进行软件测试，如按行为或结构来划分输入域的划分测试，纯粹随机选择输入的随机测试，基于功能、路径、数据流或控制流的覆盖测试等。对于给定的软件，每种测试方法都局限于暴露一定数量和一些类别的缺陷。通过这些测试能够查找、定位、改正和消除某些缺陷，实现一定意义上的软件可靠性增长。但是，由于它们都是面向错误的测试，测试所得的结果数据不能直接用于软件可靠性评价，必须经过一定的分析处理后方可使用可靠性模型进行可靠性评价。

软件可靠性测试由可靠性目标的确定、运行剖面的开发、测试用例的设计、测试实施、测试结果的分析等主要活动组成。

软件可靠性测试还必须考虑对软件开发进度和成本的影响，最好是在受控的自动测试环境下，由专业测试机构完成。

软件可靠性测试是一种有效的软件测试和软件可靠性评价技术。尽管软件可靠性测试也不能保证软件中残存的缺陷数最少，但经过软件可靠性测试可以保证软件的可靠性达到较高的要求，对于开发高可靠性与高安全性软件系统很有帮助。

软件可靠性测试要在工程上获得广泛应用，还有许多实际问题需要解决。

13.2.6　定义软件运行剖面

定义运行剖面首先需要为软件的使用行为建模，建模可以采用马尔可夫链来完成。用马尔可夫链将输入域编码为一个代表用户观点的软件使用的状态集。弧用来连接状态并表示由各种激励导致的转换，这些激励可能由硬件、人机接口或其他软件等产生。将转换概率分配给每个弧，用来代表一个典型用户最有可能施加给系统的激励。这种类型的马尔可夫链是一个离散的有限状态集，这类模型可以用有向图或转换矩阵表示。

定义运行剖面的下一步是开发使用模型，明确需要测试的内容。软件系统可能会有许多用户和用户类别，每类用户都可能以不同的方式使用系统。开发使用模型涉及到将输入域分层，有两种类型的分层形式：用户级分层和用法级分层。用户级分层依赖于谁或什么能激励系统；用法级分层依赖于在测试状态下系统能做什么。换句话说，用户级分层考虑各种类型的用户以及他们如何使用系统；用法级分层则要求考虑系统能够提供的所有功能。一旦用户和用法模型被开发出来，弧上的概率将被分配。这些概率估计主要是基于如下几个方面。

（1）从现有系统收集到的数据。

（2）与用户的交谈或对用户进行观察获得的信息。

（3）原型使用与测试分析的结果。

（4）相关领域专家的意见。

定义使用概率的最佳方法是使用实际的用户数据，如来自系统原型、前一版本的使用数据；其次是由该软件应用领域的用户和专家提供的预期使用数据；在没有任何数据可用的情况下，只能是将每个状态现有的弧分配相同的概率，这是最差的一种方法。

由于软件可靠性行为是相对于软件实际的运行剖面而言的，同一软件在不同运行剖面下其可靠性表现可能大不相同，所以用于可靠性测试准备的运行剖面的开发与定义必须充分分析和考虑软件的实际运行情况。

软件可靠性测试假设每个操作的数据输入都有同样的发生错误的概率，这样最频繁出现的操作和输入将表现出最高的故障率。对于特定的操作环境这是正确的，但无法贯穿系统的全部操作集合。典型的例子是飞机的飞行控制软件，在正常飞行、起飞、降落、地面运动和地面等待这 5 个状态中，尽管起飞和降落在运行剖面上只占有很小的百分比，但是它们却占有很大的故障比例。对于高安全性要求的软件，一个看起来很少使用的代码路径也可能带来灾难性的后果。因此，对于边界、跃迁情况和关键功能不应该用简单的运行剖面来对待，应该构造专门的运行剖面，补充统计模型之外的测试用例。在覆盖

率水平不够时，可根据具体空白，进行适当的补充测试。如果补充测试发现了错误，就可分析这些错误，估计其对可靠性产生的影响。

一个产品有可能需要开发多个运行剖面，这取决于它所包含的运行模式和关键操作，通常需要为关键操作单独定义运行剖面。

13.2.7 可靠性测试用例设计

为了对软件可靠性进行良好的预计，必须在软件的运行域上对其进行测试。首先定义一个相应的剖面来镜像运行域，然后使用这个剖面驱动测试，这样可以使测试真实地反映软件的使用情况。

由于可能的输入几乎是无限的，测试必须从中选择出一些样本，即测试用例。测试用例要能够反映实际的使用情况，反映系统的运行剖面。将统计方法运用到运行剖面开发和测试用例生成中去，并为在运行剖面中的每个元素都定量地赋予一个发生概率值和关键因子，然后根据这些因素分配测试资源，挑选和生成测试用例。

在这种测试中，优先测试那些最重要或最频繁使用的功能，释放和缓解最高级别的风险，有助于尽早发现那些对可靠性有最大影响的故障，以保证软件的按期交付。

设计测试用例就是针对特定功能或组合功能设计测试方案，并编写成文档。测试用例的选择既要有一般情况，也应有极限情况以及最大和最小的边界值情况。因为测试的目的是暴露应用软件中隐藏的缺陷，所以在设计选取测试用例和数据时要考虑那些易于发现缺陷的测试用例和数据，结合复杂的运行环境，在所有可能的输入条件和输出条件中确定测试数据，来检查应用软件是否都能产生正确的输出。

一个典型的测试用例应该包括下列组成部分。

（1）测试用例标识。

（2）被测对象。

（3）测试环境及条件。

（4）测试输入。

（5）操作步骤。

（6）预期输出。

（7）判断输出结果是否符合标准。

（8）测试对象的特殊需求。

由于可靠性测试的主要目的是评估软件系统的可靠性，因此，除了常规的测试用例集仍然适用外，还要着重考虑和可靠性密切相关的一些特殊情况。在测试中，可以考虑进行“强化输入”，即比正常输入更恶劣（合理程度的恶劣）的输入。

表 13-4 给出了一些参考例子。

表 13-4　可靠性测试用例设计时重点考虑的一些特殊情况

序　号	测 试 目 的	描　述
1	屏蔽用户操作错误	考虑对用户常见的误操作的提示和屏蔽情况
2	错误提示的准确性	对用户的错误提示准确描述
3	错误是否导致系统异常退出	有无操作错误引起系统异常退出的情况
4	数据可靠性	系统应对输入的数据进行有效性检查，对冗余的数据进行过滤、校验和清洗，保证数据的正确性和可靠性
5	异常情况的影响	考察数据和系统的受影响程度，若受损，是否提供补救工具，补救的情况如何。异常情况包括： • 硬件故障 • 网络故障 • 部分软件模块失效

13.2.8　可靠性测试的实施

在进行应用软件的可靠性测试前有必要检查软件需求与设计文档是否一致，检查软件开发过程中形成的文档的准确性、完整性以及与程序的一致性，检查所交付程序和数据以及相应的软件支持环境是否符合要求。

这些检查虽然增加了工作量，但对于在测试早期发现错误和提高软件的质量是非常必要的。

软件可靠性测试必须是受控测试，在运行此类测试时，为了保证统计数据的有效性，测试过程中的每个测试用例必须用相同的软件版本，新的软件版本意味着新测试的开始。

在有些情况下，不能进行纯粹的可靠性测试。因为客户的要求、合同的规定或者标准的约束，需要补充其他形式的非统计测试。这时的最佳选择是既执行可靠性测试，也执行非统计测试（如覆盖测试）。如果非统计测试在可靠性测试之前完成，由统计测试产生的统计数据仍然有效。但是在可靠性测试之后执行非统计测试，可能会影响软件可靠性评估的准确性。

软件可靠性测试同样依赖于软件的可测试性。可靠性测试的难点就在于判断测试用例的运行是成功还是失败。在控制系统及类似的软件中，失效由详细说明、时间（通常是 CPU 时间或时钟时间）来客观地定义。而在一般应用系统中，失效的定义更主观些，它不仅依赖于程序是否符合规格说明的要求，也取决于指定的性能是否能够达到用户的期望，但是否达到期望没有确定的标准。在一些科学计算中，计算结果只能由计算机给出，在这种情况下，如果软件只是输出了错误的结果而不是整个系统发生失效，错误就不可能被发现。此时可以将测试分成两个阶段进行。第一阶段运行较少量的测试用例，并对照规范进行仔细检查。第二阶段再运行大量测试用例。第二阶段不用人工检查输出的每项内容，而是找失效现象，包括错误信息、断电、崩溃和死机。也可把输出记录到文件中，采用搜索或过滤方法进行处理。如果软件有足够的可测试性，这种方法不会漏

掉很多的严重失效。如果计算的正确性无法验证，就需要对软件进行一些形式化的证明。

开发方交付的任何软件文档中与可靠性质量特性有关的部分、程序以及数据都应当按照需求说明和质量需求进行测试。在项目合同、需求说明书和用户文档中规定的所有配置情况下，程序和数据都必须进行测试。

软件可靠性数据是可靠性评价的基础。为了获得更多的可靠性数据，应该使用多台计算机同时运行软件，以增加累计运行时间。应该建立软件错误报告、分析与纠正措施系统。按照相关标准的要求，制定和实施软件错误报告和可靠性数据收集、保存、分析和处理的规程，完整、准确地记录软件测试阶段的软件错误报告和收集可靠性数据。

用时间定义的软件可靠性数据可以分为 4 类，具体内容如下。

（1）失效时间数据。记录发生一次失效所累积经历的时间。

（2）失效间隔时间数据。记录本次失效与上一次失效间的间隔时间。

（3）分组时间内的失效数。记录某个时间区内发生了多少次失效。

（4）分组时间的累积失效数。记录到某个区间的累积失效数。

这 4 类数据可以互相转化。

在测试过程中必须真实地进行记录，每个测试记录必须包含如下信息。

（1）测试时间。

（2）含有测试用例的测试说明或标识。

（3）所有与测试有关的测试结果，包括失效数据。

（4）测试人员。

测试活动结束后要编写《软件可靠性测试报告》，对测试用例及测试结果在测试报告中加以总结归纳。编写时可以参考 GJB 438A-97 中提供的《软件测试报告》格式，并应根据情况进行剪裁。测试报告应具备如下内容。

（1）软件产品标识。

（2）测试环境配置（硬件和软件）。

（3）测试依据。

（4）测试结果。

（5）测试问题。

（6）测试时间。

把可靠性测试过程进行规范化，有利于获得真实有效的数据，为最终得到客观的可靠性评价结果奠定基础。

13.3 软件可靠性评价

13.3.1 软件可靠性评价概述

软件可靠性评价是软件可靠性活动的重要组成部分，既适用于软件开发过程，也可

针对最终软件系统。在软件开发过程中使用软件可靠性评价，可以使用软件可靠性模型，估计软件当前的可靠性，以确认是否可以终止测试并发布软件，同时还可以预计软件要达到相应的可靠性水平所需要的时间和工作量，评价提交软件时的软件可靠性水平。对于最终软件产品，软件可靠性评价结合可靠性验证测试，确认软件的执行与需求的一致性，确定最终软件产品所达到的可靠性水平。

这一节阐述的软件可靠性评价工作是指选用或建立合适的可靠性数学模型，运用统计技术和其他手段，对软件可靠性测试和系统运行期间收集的软件失效数据进行处理，并评估和预测软件可靠性的过程。这个过程包含如下三个方面。

（1）选择可靠性模型。

（2）收集可靠性数据。

（3）可靠性评估和预测。

13.3.2　怎样选择可靠性模型

在前面讨论了软件的可靠性模型以及一个举例，一些可靠性研究者试图寻找一个最好的模型，能适用于所有的软件系统，但这样的工作是徒劳的。因为对于不同的软件系统，出于不同的可靠性分析目的，模型的适用性是不一样的。但究竟怎样来为可靠性评价选用不同的模型，却又是一个不小的难题。

针对可靠性模型的构成以及使用模型来进行可靠性评价的目的，可以从以下几个方面进行比较和选择。

1．模型假设的适用性

模型假设是可靠性模型的基础，模型假设要符合软件系统的现有状况，或与假设冲突的因素在软件系统中应该是可忽略的。例如，有的模型假定检测或发现的软件缺陷是立即排除掉的，而且排除时间忽略不计，如果现有的软件系统对于严重程度类较低的软件缺陷不进行立即排错，那么这个模型显然是不适用的。

往往一个模型的假设有许多条，我们需要在选用模型的时候对每一条假设进行细致的分析，评估现有的软件系统中不符合假设的因素对可靠性评价的影响如何，以确定模型是否适合软件系统的可靠性评价工作。

2．预测的能力与质量

预测的能力与质量是指模型根据现在和历史的可靠性数据，预测将来的可靠性和失效概率的能力，以及预测结果的准确程度。显然，模型预测的能力与质量是比较难于评价的，但任何一个模型只有在实践中加以实验和不断改善，才能得到认可。所以，在满足其他条件的前提下，应尽量选用比较成熟、应用较广的模型作为分析模型。

3．模型输出值能否满足可靠性评价需求

使用模型进行可靠性评价的最终目的，是想得到软件系统当前的可靠性定量数据，以及预测一定时间后的可靠性数据，可以根据可靠性测试目的来确定哪些模型的输出值

满足可靠性评价需求。一般来说，最重要的几个需要精确估计的可靠性定量指标包括如下内容。

（1）当前的可靠度。

（2）平均无失效时间。

（3）故障密度。

（4）期望达到规定可靠性目标的日期。

（5）达到规定的可靠性目标的成本要求。

4．模型使用的简便性

模型使用的简便性一般包含如下三层含义。

（1）模型需要的数据在软件系统中应该易于收集，而且收集需要投入的成本不能超过可靠性计划的预算。

（2）模型应该简单易懂，进行可靠性分析的软件测试人员不会花费太多的时间去研究专业的数学理论，他们只需要知道哪些假设适用，需要收集哪些数据，能够得到哪些分析结果就可以了。

（3）模型应该便于使用，最好能用工具实现数据的输入。也就是说，测试人员除了输入可靠性数据外，不需要深入模型内部进行一些额外的工作。

尽管这样，由于可靠性研究理论在软件工程领域发展的限制，可供选择的可靠性模型极其有限，这已在相当大的程度上制约了可靠性测试的开展。

13.3.3 可靠性数据的收集

面向缺陷的可靠性测试产生的测试数据经过分析后，可以得到非常有价值的可靠性数据，是可靠性评价所用数据的一个重要来源，这部分数据取决于定义的运行剖面和选取的测试用例集。可靠性数据主要是指软件失效数据，是软件可靠性评价的基础，主要是在软件测试、实施阶段收集的。在软件工程的需求、设计和开发阶段的可靠性活动，也会产生影响较大的其他可靠性数据。因此，可靠性数据的收集工作是贯穿于整个软件生命周期的。

由于软件开发过程中的特殊复杂性及许多潜在因素的影响，可靠性数据收集工作会极为困难。目前，关于数据的收集工作，存在许多有待解决的问题。

（1）可靠性数据的规范不统一，对软件进行度量的定义混乱不清。例如，时间、缺陷、失效和模型结构等的定义，就相当含糊，缺乏统一的标准。这样就使得在进行软件可靠性数据的收集时，目标不明确，甚至无从下手。

（2）数据收集工作的连续性不能保证。可靠性数据的收集是连续的、长期的过程，而且需要投入一定的资金、人力、时间，往往这些投入会在软件的开发计划中被忽略，以至于不能保证可靠性数据收集工作的正常进行。

（3）缺乏有效的数据收集手段。进行数据收集同样需要方便实用的工具，然而除了

在可靠性测试方面有了一些可用的数据收集工具外，其他方面的工具还十分缺乏。

（4）数据的完整性不能保证。即使可靠性活动计划做得再周密，收集到的数据仍有可能是不完全的，而且遗漏的数据往往会影响到可靠性评价的结果。

（5）数据质量和准确性不能保证。不完全的排错及诊断，使收集的数据中含有不少虚假的成分，它们不能正确反映软件的真实状况。使用不准确的可靠性数据进行的可靠性评价，误差有可能会比利用可靠性模型进行预测产生的误差大一个数量级，这说明数据质量的重要性。

为了给软件可靠性评价提供一套准确、有效的可靠性数据，有必要在软件工程中重视软件可靠性数据的收集工作，采取一些措施尽量解决上述问题。在现有条件下，可行的一些办法如下。

（1）及早确定所采用的可靠性模型，以确定需要收集的可靠性数据，并明确定义可靠性数据规范中的一些术语和记录方法，如时间、失效、失效严重程度类的定义，制定标准的可靠性数据记录和统计表格等。

（2）制定可实施性较强的可靠性数据收集计划，指定专人负责，抽取部分开发人员、质量保证人员、测试人员、用户业务人员参加，按照统一的规范收集记录可靠性数据。

（3）重视软件测试特别是可靠性测试产生的测试数据的整理和分析，因为这部分数据是用模拟软件实际运行环境的方法、模拟用户实际操作的测试用例测试软件系统产生的数据，对软件可靠性评价和预测有较高的实用价值。

（4）充分利用数据库来完成可靠性数据的存储和统计分析。一方面减少数据管理的混乱，一方面提高数据处理的效率。

13.3.4　软件可靠性的评估和预测

软件可靠性的评估和预测的主要目的，是为了评估软件系统的可靠性状况和预测将来一段时间的可靠性水平。下面是一些常见的需要利用软件可靠性评价进行解答的问题。

（1）判断是否达到了可靠性目标，是否达到了软件付诸使用的条件，是否达到了中止测试的条件。

（2）如未能达到，要再投入多少时间、多少人力和多少资金，才能达到可靠性目标或投入使用。

（3）在软件系统投入实际运行一年或若干时间后，经过维护、升级、修改，软件能否达到交付或部分交付用户使用的可靠性水平。

目前有不少支持软件可靠性估计的软件工具，只要将收集的失效数据分类并录入，选择合适的可靠性模型就可以获得软件可靠性的评价结果。

然而，对于那些可靠性要求很高的系统，必须进行很多测试才能预计出高置信度的可靠性结果。即便如此，仍然可能没有任何失效发生。没有失效就无法估计可靠性，不能认为程序的可靠度是 1.0。除非已经进行了完全的测试，否则程序不失效就无法做出估

计，而完全的测试几乎总是不可能的。如果在测试期间没有失效发生，可以简单地假设测试是基于二项式分布的，这样就可以对可靠性作保守估计。也可以凭经验，根据无故障运行的测试用例的数量，在一定的置信度水平上，估计可靠性的等级。

软件可靠性评价技术和方法主要依据选用的软件可靠性模型，其来源于统计理论。软件可靠性评估和预测以软件可靠性模型分析为主，但也要在模型之外运用一些统计技术和手段对可靠性数据进行分析，作为可靠性模型的补充、完善和修正。这些辅助方法如下。

（1）失效数据的图形分析法。运行图形处理软件失效数据，可以直观地帮助我们进行分析。图形指标如下。

① 累积失效个数图形。

② 单位时间段内的失效数的图形。

③ 失效间隔时间图形。

（2）试探性数据分析技术（Exploratory Data Analysis，EDA）。对于失效数据图形进行一定的数字化分析，能发现和揭示出数据中的异常。对可靠性分析有用的信息如下。

① 循环相关。

② 短期内失效数的急剧上升。

③ 失效数集中的时间段。

这种分析方法常可以发现因排错引入新的缺陷、数据收集的质量问题及时间域的错误定义等问题。

还有其他一些分析方法，这里就不一一赘述了。

13.4 软件的可靠性设计与管理

13.4.1 软件可靠性设计

在测试阶段，利用测试手段收集测试数据，并利用软件可靠性模型，可以评估或预测软件的可靠性。这些软件可靠性测试活动虽然能通过查错—排错活动有限地改善软件可靠性，但不能从根本上提高软件的可靠性，也难以保证软件可靠性，并且修改由于设计导致的软件缺陷，有可能付出比较昂贵的代价。实践证明，保障软件可靠性最有效、最经济、最重要的手段是在软件设计阶段采取措施进行可靠性控制。为了从根本上提高软件的可靠性，降低软件后期修改的成本和难度，人们提出了可靠性设计的概念。

可靠性设计其实就是在常规的软件设计中，应用各种方法和技术，使程序设计在兼顾用户的功能和性能需求的同时，全面满足软件的可靠性要求，即采用一些技术手段，把可靠性“设计”到软件中去。软件可靠性设计技术就是以提高和保障软件的可靠性为目的，在软件设计阶段运用的一种特殊的设计技术。

在软件工程中已有很多比较成熟的设计技术，如结构化设计、模块化设计、自顶向下设计及自底向上设计等，这些技术是为了保障软件的整体质量而采用的。在此基础上，为了进一步提高软件的可靠性，通常会采用一些特殊设计技术。虽然软件可靠性设计技术与普通的软件设计技术没有明显的界限，但软件可靠性设计仍要遵循一些自己的原则。

（1）软件可靠性设计是软件设计的一部分，必须在软件的总体设计框架中使用，并且不能与其他设计原则相冲突。

（2）软件可靠性设计在满足提高软件质量要求的前提下，以提高和保障软件可靠性为最终目标。

（3）软件可靠性设计应确定软件的可靠性目标，不能无限扩大化，并且排在功能度、用户需求和开发费用之后考虑。

可靠性设计概念被广为引用，但并没有多少人能提出非常实用并且广泛运用的可靠性设计技术。一般来说，被认可的且具有应用前景的软件可靠性设计技术主要有容错设计、检错设计和降低复杂度设计等技术。

1．容错设计技术

对于软件失效后果特别严重的场合，如飞机的飞行控制系统、空中交通管制系统及核反应堆安全控制系统等，可采用容错设计方法。常用的软件容错技术主要有恢复块设计、N 版本程序设计和冗余设计三种方法。

（1）恢复块设计。

程序的执行过程可以看成是由一系列操作构成的，这些操作又可由更小的操作构成。恢复块设计就是选择一组操作作为容错设计单元，从而把普通的程序块变成恢复块。被选择用来构造恢复块的程序块可以是模块、过程、子程序和程序段等。

一个恢复块包含有若干个功能相同、设计差异的程序块文本，每一时刻有一个文本处于运行状态。一旦该文本出现故障，则用备份文本加以替换，从而构成“动态冗余”。软件容错的恢复块方法就是使软件包含有一系列恢复块。

（2）N 版本程序设计。

N 版本程序的核心是通过设计出多个模块或不同版本，对于相同初始条件和相同输入的操作结果，实行多数表决，防止其中某一软件模块/版本的故障提供错误的服务，以实现软件容错。为使此种容错技术具有良好的结果，必须注意以下两个方面。

① 使软件的需求说明具有完全性和精确性。这是保证软件设计错误不相关的前提，因为软件的需求说明是不同设计组织和人员的唯一共同出发点。

② 设计全过程的不相关性。它要求各个不同的软件设计人员彼此不交流，程序设计使用不同的算法、不同的编程语言、不同的编译程序、不同的设计工具、不同的实现方法和不同的测试方法。为了彻底保证软件设计的不相关性，甚至提出设计人员应具有不同的受教育背景，来自不同的地域、不同的国家。

（3）冗余设计。

改善软件可靠性的一个重要技术是冗余设计。在硬件系统中，在主运行的系统之外备用额外的元件或系统，如果出现一个元件故障或系统故障，则立即更换冗余的元件或切换到冗余的系统，则该硬件系统仍可以维持运行。在软件系统中，冗余技术的运用有所区别。如果采用相同两套软件系统作为互为备份，其意义不大，因为在相同的运行环境中，一套软件出故障的地方，另外一套也一定会出现故障。软件的冗余设计技术实现的原理是在一套完整的软件系统之外，设计一种不同路径、不同算法或不同实现方法的模块或系统作为备份，在出现故障时可以使用冗余的部分进行替换，从而维持软件系统的正常运行。

从表面上看，设计开发完成同样功能但实现方法完全不同的两套软件系统，需要的费用可能接近于单个版本软件开发费用的两倍，但采用冗余技术设计软件所增加的额外费用肯定远低于重新设计一个版本软件的费用。这是因为大多数设计花费，例如文档、测试以及人力都是有可能复用的。冗余设计还有可能导致软件运行时所花费的存储空间、内存消耗以及运行时间有所增加，这就需要在可靠性要求和额外付出代价之间作出折衷。

2．检错技术

在软件系统中，对无需在线容错的地方、或不能采用冗余设计技术的部分，如果对可靠性要求较高，故障有可能导致严重的后果。一般采用检错技术，在软件出现故障后能及时发现并报警，提醒维护人员进行处理。检错技术实现的代价一般低于容错技术和冗余技术，但它有一个明显的缺点，就是不能自动解决故障，出现故障后如果不进行人工干预，将最终导致软件系统不能正常运行。

采用检错设计技术要着重考虑几个要素：检测对象、检测延时、实现方式和处理方式。

（1）检测对象：包含两个层次的含义，即检测点和检测内容。在设计时应考虑把检测点放在容易出错的地方和出错对软件系统影响较大的地方；检测内容选取那些有代表性的、易于判断的指标。

（2）检测延时：从软件发生故障到被自检出来是有一定延时的，这段延时的长短对故障的处理是非常重要的。因此，在软件检错设计时要充分考虑到检测延时。如果延时长到影响故障的及时报警，则需要更换检测对象或检测方式。

（3）实现方式：最直接的一种实现方式是判断返回结果，如果返回结果超出正常范围，则进行异常处理。计算运行时间也是一种常用的技术，如果某个模块或函数运行超过预期的时间，可以判断出现故障。另外，还有置状态标志位等多种方法，自检的实现方式要根据实际情况来选用。

（4）处理方式：大多数检错采用“查出故障—停止软件系统运行—报警”的处理方式，但也有采用不停止或部分停止软件系统运行的情况，这一般由故障是否需要实时处理来决定。

3．降低复杂度设计

前面讲到，软件和硬件最大的区别之一就是软件的内部结构比硬件复杂得多，我们用软件复杂度来定量描述软件的复杂程度。软件复杂性常分为模块复杂性和结构复杂性。模块复杂性主要包含模块内部数据流向和程序长度两个方面，结构复杂性用不同模块之间的关联程度来表示。软件复杂度可用涉及到模块复杂性和结构复杂性的一些统计指标来进行定量描述，在这里就不进行详细叙述了。

软件的复杂性与软件可靠性有着密切的关系，软件复杂性是产生软件缺陷的重要根源。有研究表明，当软件的复杂度超过一定界限时，软件缺陷数会急剧上升，软件的可靠性急剧下降。因此，在设计时就应考虑降低软件的复杂性，使之处于一个合理的阀值之内，这是提高软件可靠性的有效方法。

降低复杂度设计的思想就是在保证实现软件功能的基础上，简化软件结构，缩短程序代码长度，优化软件数据流向，降低软件复杂度，从而提高软件可靠性。

除了容错设计、检错设计和降低复杂度设计技术外，人们尝试着把硬件可靠性设计中比较成熟的技术，如故障树分析（FTA）、失效模式与效应分析（FMEA）等运用到软件可靠性设计领域，这些技术大多是运用一些分析、预测技术，在软件设计时就充分考虑影响软件可靠性的因素，并采取一些措施进行优化。由于软件与硬件内部性质的巨大差异，这些技术在软件可靠性设计领域的应用效果和范围极其有限。

13.4.2　软件可靠性管理

为了进一步提高软件可靠性，人们又提出了软件可靠性管理的概念，把软件可靠性活动贯穿于软件开发的全过程。

软件可靠性管理是软件工程管理的一部分，它以全面提高和保证软件可靠性为目标，以软件可靠性活动为主要对象，是把现代管理理论用于软件生命周期中的可靠性保障活动的一种管理形式。

软件可靠性管理的内容包括软件工程各个阶段的可靠性活动的目标、计划、进度、任务和修正措施等。

软件工程各个阶段可能进行的主要的软件可靠性活动如下所述。由于软件之间的差异较大，并且人们对可靠性的期望不同，对可靠性的投入不同，所以下面的每项活动并不是每一个软件系统的可靠性管理的必须内容，也不是软件可靠性管理的全部内容。

1．需求分析阶段

（1）确定软件的可靠性目标。

（2）分析可能影响可靠性的因素。

（3）确定可靠性的验收标准。

（4）制定可靠性管理框架。

（5）制定可靠性文档编写规范。

（6）制定可靠性活动初步计划。

（7）确定可靠性数据收集规范。

2．概要设计阶段

（1）确定可靠性度量。

（2）制定详细的可靠性验收方案。

（3）可靠性设计。

（4）收集可靠性数据。

（5）调整可靠性活动计划。

（6）明确后续阶段的可靠性活动的详细计划。

（7）编制可靠性文档。

3．详细设计阶段

（1）可靠性设计。

（2）可靠性预测（确定可靠性度量估计值）。

（3）调整可靠性活动计划。

（4）收集可靠性数据。

（5）明确后续阶段的可靠性活动的详细计划。

（6）编制可靠性文档。

4．编码阶段

（1）可靠性测试（含于单元测试）。

（2）排错。

（3）调整可靠性活动计划。

（4）收集可靠性数据。

（5）明确后续阶段的可靠性活动的详细计划。

（6）编制可靠性文档。

5．测试阶段

（1）可靠性测试〔含于集成测试、系统测试〕。

（2）排错。

（3）可靠性建模。

（4）可靠性评价。

（5）调整可靠性活动计划。

（6）收集可靠性数据。

（7）明确后续阶段的可靠性活动的详细计划。

（8）编制可靠性文档。

6．实施阶段

（1）可靠性测试（含于验收测试）。

（2）排错。

（3）收集可靠性数据。

（4）调整可靠性模型。

（5）可靠性评价。

（6）编制可靠性文档。

可靠性管理目前还停留在定性描述的水平上，很难用量化的指标来进行可靠性管理。可靠性管理规范的制定水平和实施效果也有待提高。怎样利用有限的可靠性投入，达到预期的可靠性目标是软件项目管理者常常要面对的难题。因此，可靠性管理研究是一个长期的课题。

第 14 章　基于 ODP 的架构师实践

软件系统架构设计方法是一个实践性大于理论性的工作。从软件有模块概念那天起，就有了总体设计，研究模块、构件与它们之间的关系。架构设计虽然可以归集到几种风格，但面对复杂的应用环境，不同应用领域对架构的理解差异非常大，用事实说话是最基本的研究方法。本章在 RM-ODP 多视点架构模型上，探讨应用于分布式信息系统的软件架构开发，对软件生命周期其他阶段的影响，特别是架构师在开发过程中的任务与作用。

14.1　基于 ODP 的架构开发过程

系统架构反映了功能在系统构件中的分布、基础设施相关技术和架构设计模式等，它包含了架构的原则和方法、构件关系与约束，并能支持迭加或增量开发。以软件架构为中心的开发过程是以质量和风险驱动的，最终提供一个稳定、低风险的系统架构，并满足客户的需求（包含潜在需求）。

开放分布进程的参考模型（RM-ODP）是一个 ISO 标准，它为分布式计算进程提供了一个框架。RM-ODP 定义了分布式系统的重要性质：开放性、整体性、灵活性、可塑性、联合性、可操作管理性、优质服务、安全性和透明性，并定义了一组视点。RM-ODP 视点定义大体对应于 IEEE 1471 定义，RM-ODP 定义的 5 个视点如下。

（1）企业视点：在如下因素的环境中分析系统，商业需求和策略、以及系统的范围和目的。RM-ODP 处理可能会影响系统中的与企业相关的信息，如组织结构等。

（2）信息视点：指信息的结构，它的变化、流程以及在不同功能间的逻辑划分。

（3）计算视点：重点在于把系统分解为实体和实体间的接口。

（4）工程视点：处理分布式系统对象之间的交互，以及交互是如何得到支持的。

（5）技术视点：定义构成系统的硬件和软件构件。

体系结构视点是把抽象的符号或图表（如 UML）运用到具体的体系结构开发任务中。每一个视点有具体的建模目标和系统相关者。例如，环境视图提供了对系统边界及与系统发生交互的外部实体集合的概述。分析视图提供了一个以建模问题而不是答案为中心的实体的抽象集合。

以描述软件设计为目的的视点包括构件、构件交互及构件状态。视图提供了一个对于逻辑运行结构及其功能，以及它们之间通信的映射。子系统接口依赖视图提供了一个子系统依赖关系和接口的图形表示；分层子系统视图提供了一个所有子系统高度抽象的

视图；逻辑数据视图提供了构件共有的数据模型描述。

不同视图解决不同方面的问题，这是应对复杂问题的基本研究方法（分治）。采用 ODP 从 5 个视点描述信息系统架构，对整个系统开发过程有一定指导意义。除了架构设计阶段，其他阶段对架构师也提出不同的任务与要求。图 14-1 展示了整个系统及架构开发的 10 个过程。

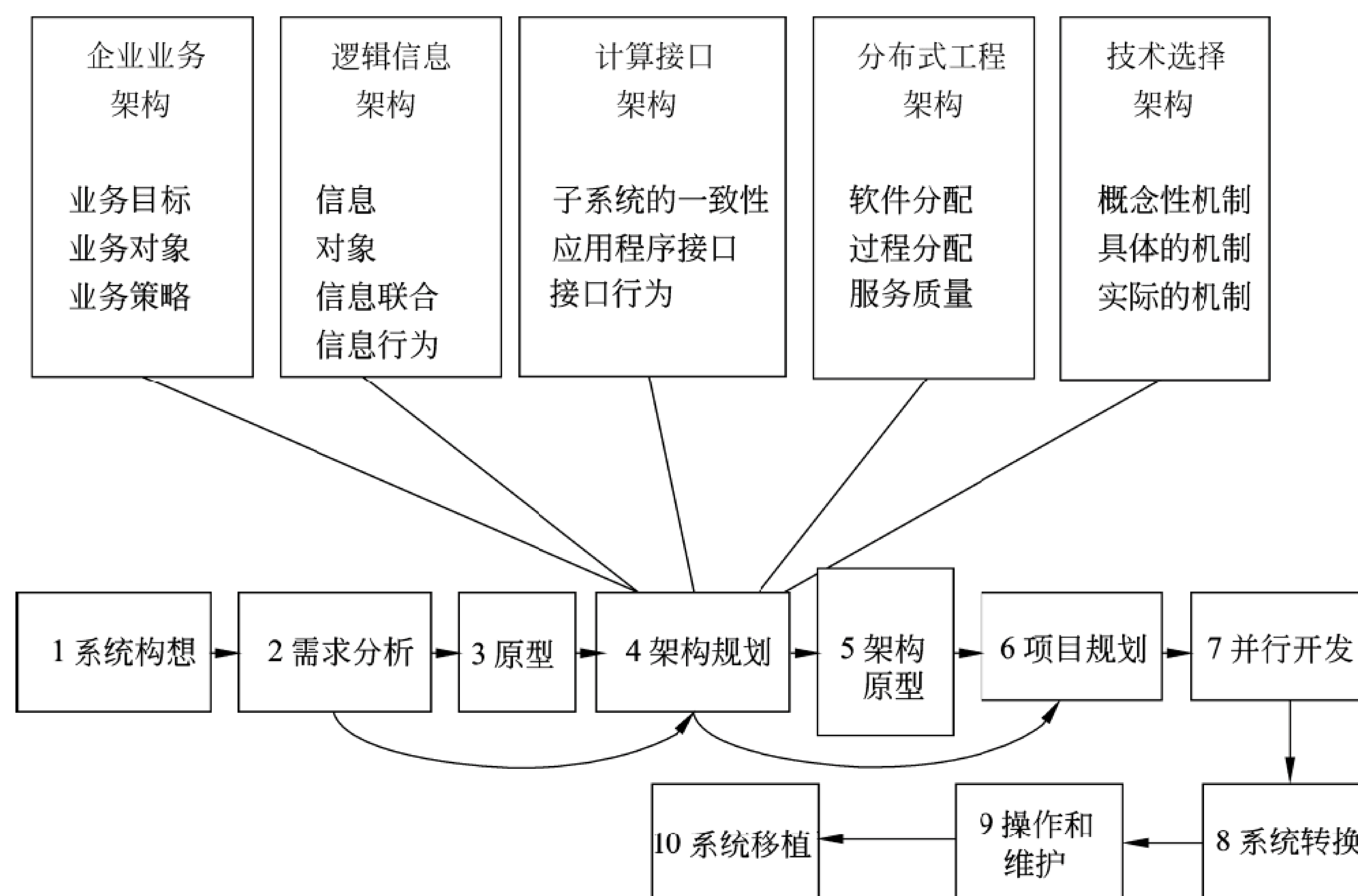


图 14-1　系统架构开发的 10 个过程

本章按图 14-1 的次序，探讨架构师的任务与设计工作。

14.2　系统构想

14.2.1　系统构想的定义

系统构想是指一个系统开发人员与系统用户之间共同的协议。按照该协议，系统开发人员需在特定的时间内完成系统用户的需求。系统构想必须简短而切中要点，给人以清晰的感觉。它不是一成不变的，必须根据系统的不同而不同。

构想描述建立了从需求分析开始的所有项目活动的语境，它高度概括了企业业务架构的核心内容。

14.2.2 架构师的作用

讨论建模的时候，我们曾提到关键词有目的、关注点、假设和优先级，它们都是系统级的“构想描述（Vision Statement）”的基本元素。如果它们在系统开发过程中改变，项目就有被抛弃的危险。因此，以架构为中心的开发的第一步就是建立一个构想描述，且假定构想描述在系统的各个开发阶段不会改变。所有的改变必须在关键的项目计划中有所反映，特别是在系统架构中。

系统构想包括为客户、为软件系统开发团队等受益人创建的，有助于各方明了系统的目标和范围。对开发者而言，从宏观层面上显示系统架构的需求，为待开发系统提供一个结构清晰的概要，确保系统开发的计划、设计等阶段能依次有序地展开。

系统构想阶段，架构师合理的介入，有以下好处。

（1）有利于使系统架构师本身对系统的看法更加全面、准确。

（2）有利于统一系统开发人员对系统的看法。

（3）有利于正确确定需求的优先次序。

（4）通过系统构想，可以在最大程度上提高客户对设计等过程的参与程度，更好地与客户沟通。

14.2.3 系统构想面临的挑战

建立和共享架构构想要面临着很多的挑战：架构师对其控制能力之外的因素（例如组织等）通常无能为力；当产品线由一个架构来支撑时，构想就会受更多的因素制约。此外，如果共享的架构构想有问题时，不易马上觉察到。不过，可以通过有效地评估，以及高级经理和架构师之间保持紧密的联系来克服这些困难。

除了以上介绍的挑战外，在系统构想阶段，还必须面对以下几种情况。

（1）很多架构师把架构看成是他们独自的创造，而且只要他们认为合适的就进行修改。

（2）有些人不是拥有产品线构想的高级经理，却总是由这些人来决定雇佣谁来做架构师。由于没有参与架构师的招聘工作，高级经理们将无法评估架构师的能力以及理解并实现其构想。

14.3 需求分析

14.3.1 架构师的工作

需求一般定义系统的外部行为和外观及用户信息，而不用设计系统的内部结构。外部行为包括了用来保证外部行为能够完成而所需的内部行为（例如持续性或计算）。外观包括用户界面的布局和导航，用户信息包含用户概念数据结构及关系模型。

架构师对需求分析通常考察以下 6 个方面的内容。

（1）系统范围对象关系图。主要用于定义系统与系统外部实体间的界限和接口的简单模型，它可以为需求确定一个范围。

（2）用户接口原型。可将其看作为用户操作的一个雏形，通过该接口界面用户能够用一系列的操作完成它想达到的效果。

（3）需求的适用性。即这个需求应该用什么技术解决，它实现后的性能怎么样，是否与其他需求相重合或是矛盾。需求分析应注重需求本身的实用或适用，而不必考虑其实现。

（4）确定需求的优先级。可采用迭代周期来说明何时完成。

（5）为需求建立功能结构模型。可以用 UML 创建组件图和实体数据对象图，概述系统原型。

（6）使用质量功能分配（Quality Function Deployment，QFD）。根据需求的理解发现隐藏质量需求，建立相关质量场景和易变需求场景，先期预测需求风险。

架构师的一个有效地捕捉行为需求的方法是分析用例（use case）。一个用例包含一个顶层的图和扩展的文字描述。用例符号简单、抽象，非常适合于用来保证在表述顶层需求概念时的简单性和清晰度。

14.3.2　需求分析的任务

1．需求分析的目的

需求分析的目的是完整、准确地描述用户对系统的需求，跟踪用户需求的变化。将用户的需求准确地反映到系统的架构和设计中，设计和用户的需求保持一致。需求分析具有决策性、方向性和策略性的作用，它在软件开发的过程中具有举足轻重的地位。

2．需求分析的特点

一般来说，需求分析特点的共同点都是追求系统需求的完整性、一致性和验证性。

（1）完整性：是准确、全面地描述用户对系统架构的需求。

（2）一致性：是通过分析整理，剔除用户需求矛盾的方面，规范用户需求。

（3）验证性：是需求的一致性表现形式，主要包含以下几个方面的含义。

① 保持和用户要求的同步。

② 保持需求分析各侧面之间的一致。

③ 保持需求和系统设计间的同步。

因此，在对系统架构需求分析之前必须建立需求分析技术层面的基本框架，从技术上保证需求分析的要求，在此基础上进行的架构需求分析才能满足项目对需求分析的要求。

14.3.3　需求文档与架构

每个用例都有一个相关需求的文字描述。这种方法采用了包含一系列活动的列表形式，用特定领域的平铺直叙的文字来描述。定义用例应该和领域专家一起进行，如果没有领域专家的长期参与，这种活动只能是一种“伪分析”。

用例为定义架构提供了一个系统的领域行为模型。在开发的第 7 个过程中，用例被

特定系统的场景所扩展，最后这些场景会在软件测试中得到运用。

用户界面的外观、功能和导航同用例紧密相联。一个有效定义屏幕的方法叫做低保真度原型（Low-fidelity Prototyping）。 在这种方法中，屏幕是用纸和笔先画出来的。同样，最终用户领域专家也始终参与到屏幕定义中去。

有了用例和定义的用户界面以及领域概念模型，我们建立了架构规划的环境。在产生文档之外（包括纸、笔的草图），架构小组得到最终用户领域中需求功能的更深刻理解。需求分析的项目词汇表，也将在架构规划中被扩展。

14.4 系统架构设计

系统架构沟通了需求和软件之间巨大的语义上的鸿沟。需求是模糊的、直观的，而软件则具有相反的性质。系统架构的第一个任务就是定义这两个极端之间的映射，架构用一种更为技术性的方式来捕捉直觉的决定，它在设计和编码之前定义了内部的系统结构。架构设计同时为项目计划服务，它允许系统构建用适应变化的方法来控制复杂性，同时指导建立软件项目与架构对应的组织。

开放分布式处理（Open Distributed Processing，ODP）从 5 个标准的视点组织分析了系统的架构，描述了同一系统的重要方面。如图 14-2 所示，这些视点包括企业、逻辑信息、计算接口、分布式工程和技术选择。对于每个视点，确认架构需求的一致性是非常重要的。ODP 促进了这个过程，因为它内嵌了一个普遍的一致性方法，简单的一致性清单包含识别架构中一致点所需的全部内容。

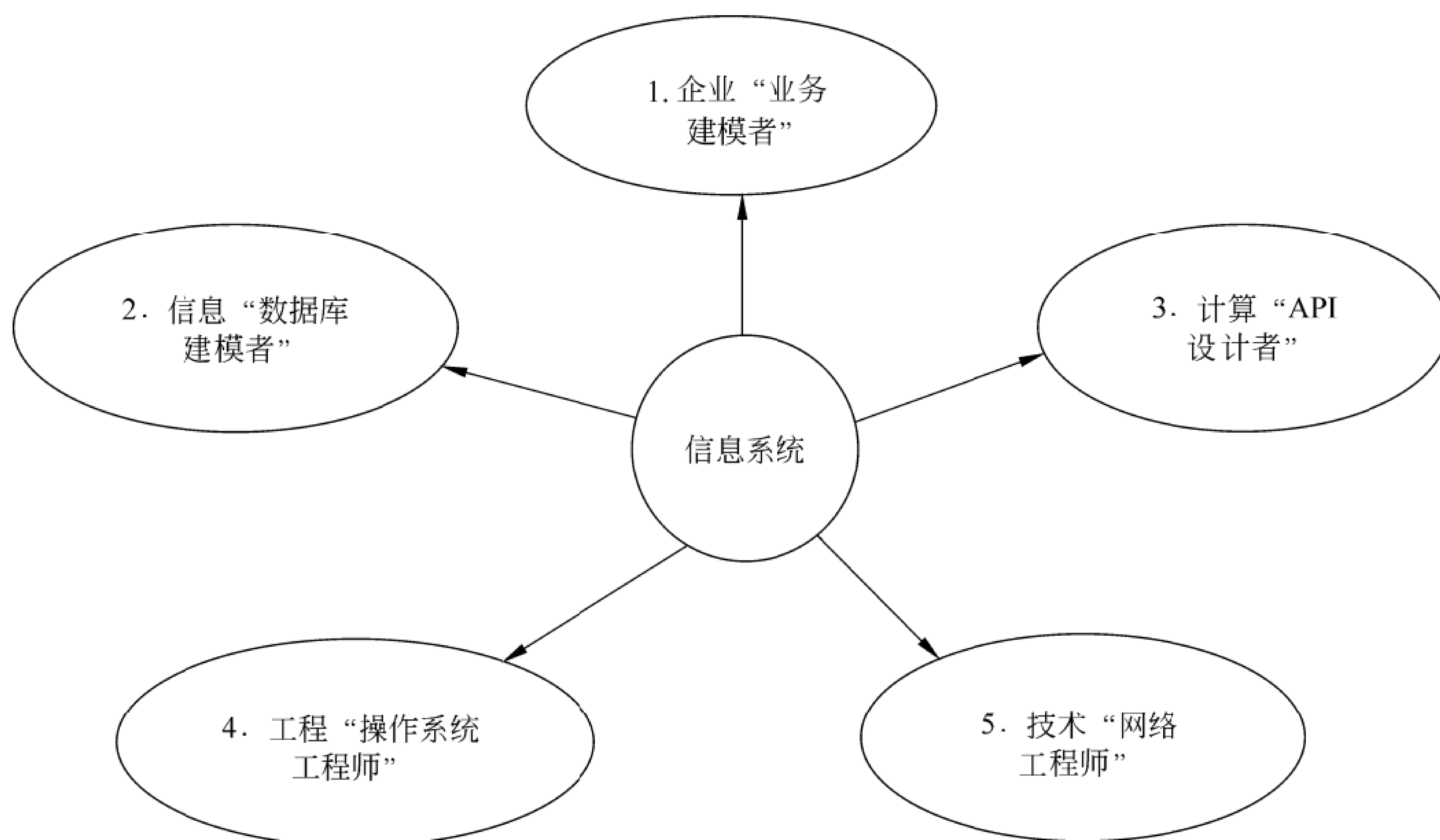


图 14-2 ODP 视点

14.4.1　企业业务架构

企业业务架构从 IT 的角度，对企业的业务结构、企业机构与业务的关系、企业内部的关系以及企业与外部机构的关系进行整理定义。企业业务架构包含如下内容。

（1）企业的业务和战略目标。描述企业的目标，包含近期目标、中期目标和长远的战略目标。

（2）企业的组织机构。明确描述企业的组织机构和职能，以及与企业相关的机构和个体，如客户、合作伙伴和供应商等。

（3）业务的分类。对企业的产品、服务和资源体系进行分类。这种分类包含了对相关产品、服务和资源的共性提取和总结。

（4）各类业务之间的关系。对产品、服务和资源的相互关联进行总结。业务之间的关系体现为跨业务的流程及资源共享等。

（5）组织机构与业务的关系。业务的执行是由机构来完成的，但是机构与业务并不一定是一一对应的关系。清楚地找出机构与业务的关系，将为应用与集成架构奠定可靠的基础。

（6）企业与外部机构的关系。对与企业相关的外部机构或个人就其类型、业务类别和业务往来模式等进行分类。

企业业务架构（企业视点）也是用高层企业对象来定义业务目的和系统策略。这些业务对象模型标识出系统的关键性约束，其中包括系统目标和重要的系统策略。

策略包含如下三类明确的表达方式。

- 责任：业务对象必须做什么。
- 许可：业务对象可以做什么。
- 禁止：业务对象不可以作什么。

在对业务问题进行分析时，不仅要考虑企业目前业务的情况，而且要考虑企业业务的发展，如新的服务或产品的推出、考虑组织机构的改变等，企业的业务流程的变化也是要考虑的因素。所有这些可能的变化（易变场景）都应该体现在企业的业务架构中。

企业业务架构在明确了企业的业务和战略目标之后，从业务和机构两个基本点出发进行基础性的分类组织工作，然后根据业务的分工和业务流程与组织机构实现映射，从而形成对企业业务的完整描述。一个典型的企业业务架构包含一系列逻辑对象图（通常用 UML 表示）和对象语义的平铺直叙的文字描述。

通过对企业业务架构的定义，就可以很清楚地知道由于企业业务特点、业务流程的特点和企业的组织机构等原因对 IT 系统所带来的自然分块和各个分块之间的边界关系，从而就可以知道怎样从技术架构上来满足和支持企业的业务架构。

企业业务架构的维护也是一个长期而反复的工作。企业业务架构的变化可以通过技术架构反映出来，技术架构的正确与否可以通过业务架构来检验，这样才能通过架构来保证 IT 服务于企业的业务和战略。

下面以一个测试结果报告系统（Test Results Reporting System，TRRS）为例，介绍一下它的企业业务架构。

TRRS 的企业视点由一些 UML 用例组成，这些用例确定了 TRRS 社区的参与者以及他们之间策略上的联系。图 14-3 展示了这些来自应用软件开发者视点的 UML 用例。这三个在 UML 图中的用例表明，软件开发者可以通过多种途径使用 TRRS，以决定软件产品的兼容性。

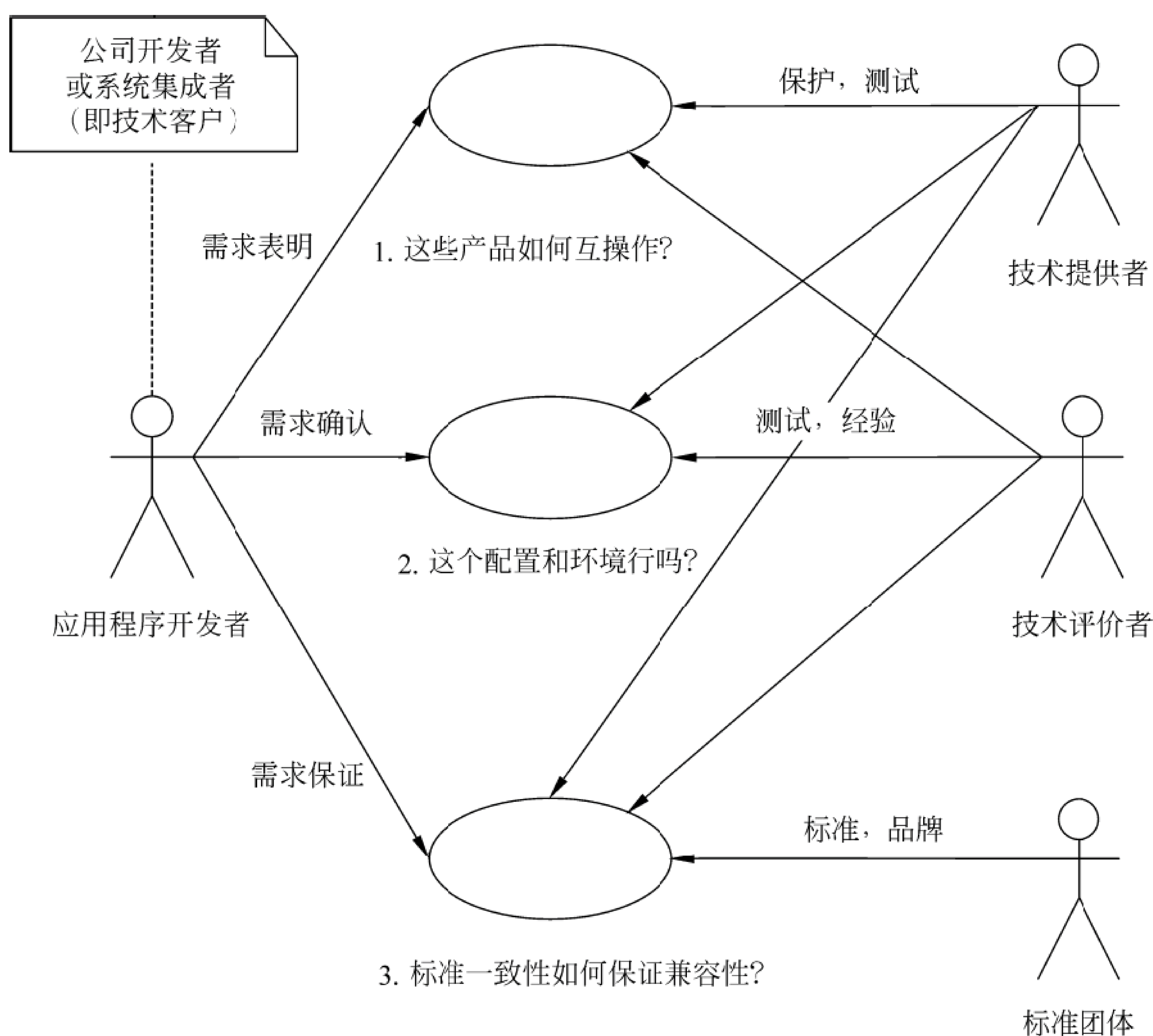


图 14-3 TRRS 企业业务架构

重要的企业策略关系到 TRRS 数据库中产品描述的完整性和责任。在 TRRS 处理中，可以使用 UML 对象约束语言（Object Constraint Language，OCL）来定义企业活动者的这些策略（如许可、禁止和义务等）。

14.4.2 逻辑信息架构

逻辑信息架构（信息视点）标识出系统必须知道什么。这种架构通过一个对象模型来表达，强调定义系统状态的属性。因为开放分布式处理是一种面向对象的方法，模型包含了关键信息的处理，如传统的对象概念。

软件架构对象并不是编程的对象，它表示对系统的约束和依赖。这些约束能够消除在把需求翻译成软件过程中的许多猜测性工作。架构师应该把他们的建模集中于系统中有高风险、高复杂性和模糊性的关键方面，而把直接的细节放在开发的环节中去。

下面以测试结果报告系统为例，介绍一下它的逻辑信息架构。

TRRS 信息视点是由一组 UML 类模型组成，该信息视点定义了一些核心的概念，这些概念组成了 TRRS 系统的持久状态。图 14-4 是一个 UML 图，它展示了产品之间的互操作关系。一致性声明（Conformance Statements，如图 14-5 所示）提供了产品兼容性

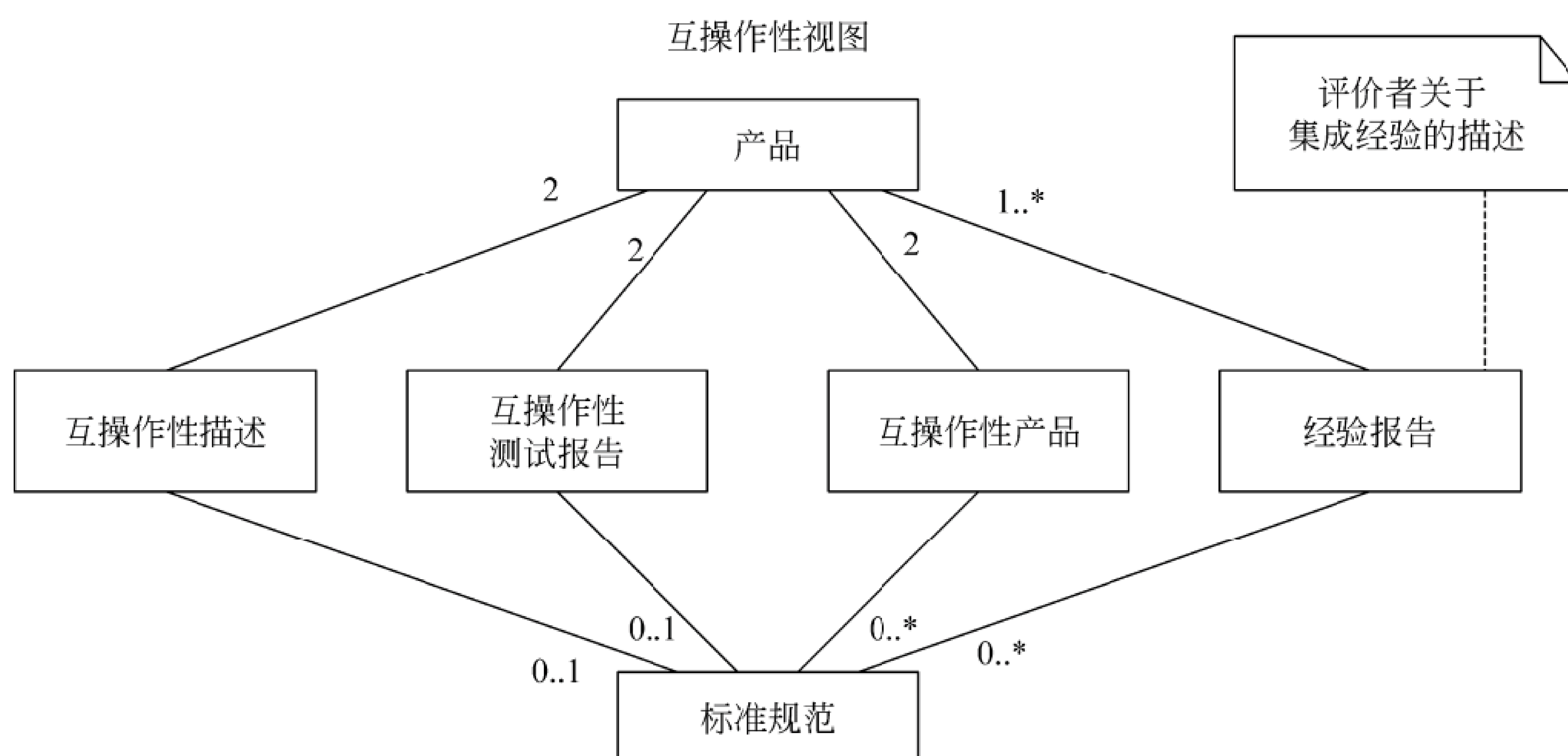


图 14-4　产品信息的 UML 表示

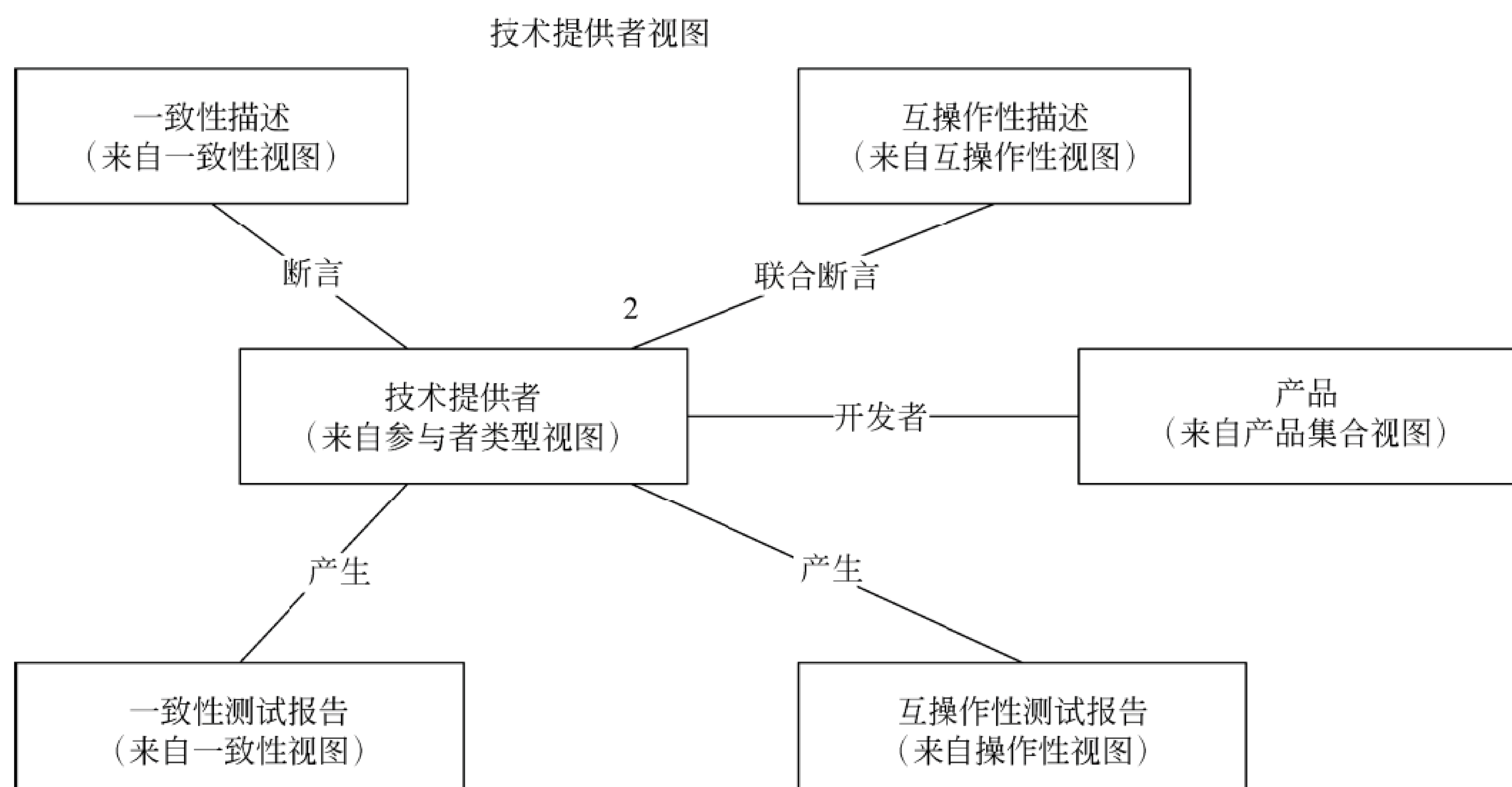


图 14-5　供应商信息的 UML 表示

标准的保证。互操作性声明（Interoperability Statements）是一个类似的概念，和兼容性不同之处在于它不包含供应商对相互之间产品兼容性的保证。互操作性测试报告（Interoperability Test Report）包括了多产品互操作测试所得出的测试结果。互操作性产品（Interoperability Product）是特定的源于多供应商兼容性的解决方案。经验报告（Experience Report）是实例研究的文档，它记载了产品集成的成功经验。合起来，上述各个部分组成了 TRRS 数据库要储存的关键文档类型。

14.4.3 计算接口架构

计算接口对系统架构非常有帮助，但是它常常被架构师所忽略。它定义了顶层的应用程序接口，这些是完全工程化的子系统边界的接口。在实现时，开发者将对他们的模型在这些边界上进行编程，以消除多个开发者和小组的主要设计争端。这些接口的架构控制对于一个支持变化和控制复杂性的稳定的系统结构来说，是非常重要的。

开放分布式处理体系结构的一个 ISO 标准采用的是 CORBA 接口定义语言（IDL），IDL 是一种基本记法，它完全独立于编程语言和操作系统。IDL 可以被编译器自动翻译成 Java、C++和 C# 等大多数流行的编程语言。

14.4.4 分布式工程架构

分布式工程架构定义了底层结构的需求，而独立于所选择的技术。它很好地解决了一些最复杂的系统策略，其中包括物理位置、系统规模可变性和通信服务质量。

ODP 的一个最大好处是关注点分离，幸运的是，前面的视点解决了许多其他的复杂问题，那些是分布式很少关注的，如 API、系统策略和信息纲要。相反，这些其他的视点能够解决它们各自的设计要点，而独立于分布式的考虑。

在进行分布式工程架构建模时，必须考虑系统的各个方面，如对象复制、多线程和系统拓扑等。

14.4.5 技术选择架构

技术选择架构（技术视点）确定了实际的技术选择，所有其他视点都独立于这些决定。因为大多数架构设计是独立的，商业技术的发展可以很容易地适应。

一个系统的选择过程包括初始的概念性机制的确认，如持久性或者通信。概念性机制的特定属性可以从其他视点得到。具体的机制被标识出来，如 DBMS、OODBMS。这些特定的参选产品是从可得到的技术中选出来的。基于对候选者的初始选择，这个过程根据产品价格、培训要求和维护风险之类的项目因素而反复进行。

架构师选择的原因是非常重要的，因为所有这些观点可以作为以后架构约束的理由。记录可以放在一个由架构小组维护的非正式项目记事本上，可以用于以后进行参考。

以测试结果报告系统为例，介绍一下它的技术选择架构。

TRRS 技术视点包括了原型规划的三种方式（如图 14-6 所示）。我们经常选用这些原型来支持渐进的系统演化和可扩展性。而从一种方式到另一种方式的演化之所以能够发生，是由在实现时选用不同的技术和提供多层结构间互操作机制所造成的。

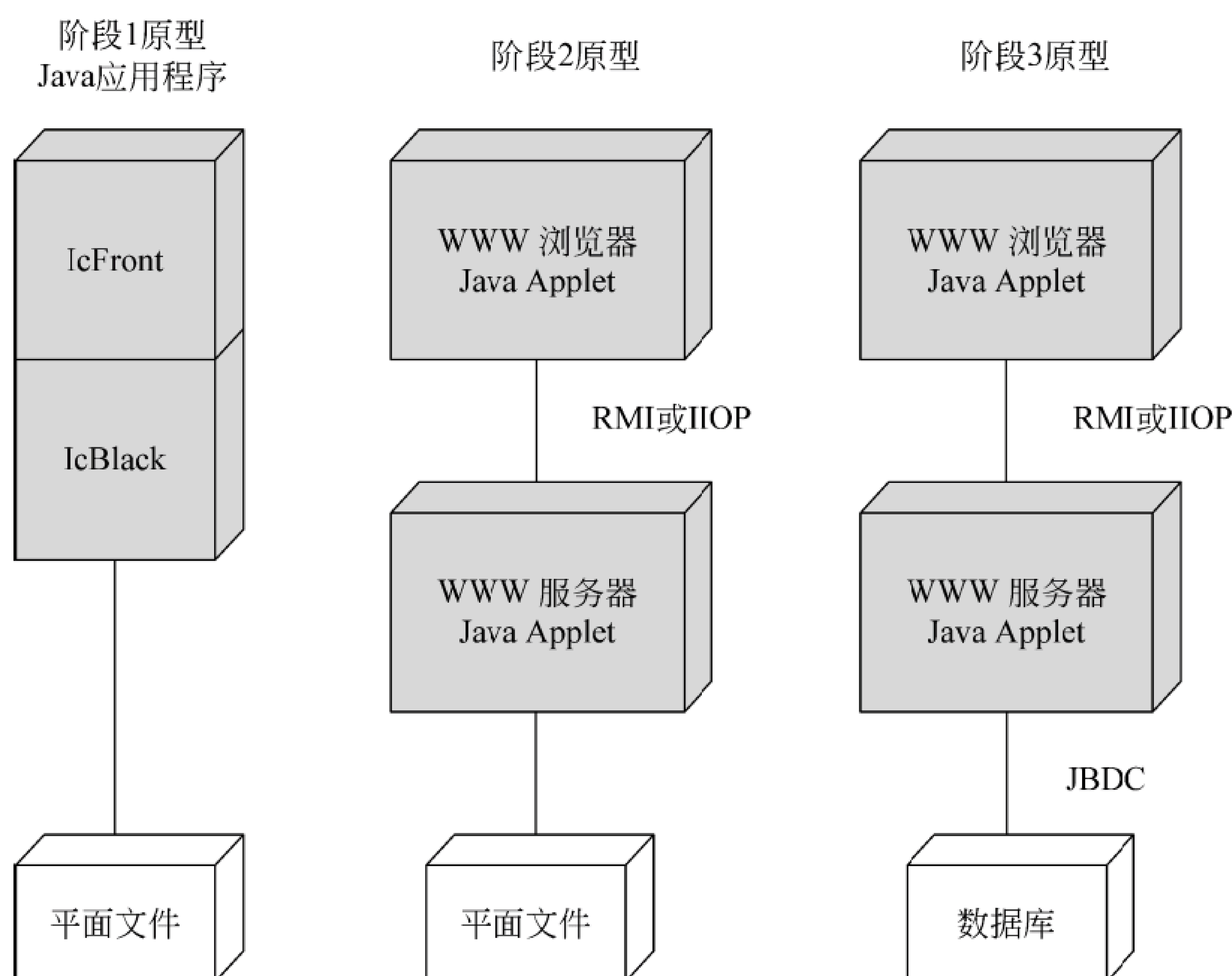


图 14-6　原型规划的三种方式

阶段 1 是一种快速的原型，它由一个独立的 Java 应用程序以及一个平面文件数据库配置而成。阶段 2 使用分布式基础设施中的 RMI 或者 IIOP 技术，支持局域网上的多客户系统。阶段 3 支持数据库的可扩展性，这是通过把平面文件替换为 JDBC 接口及其操作的后端数据库来实现的。

在阶段 3 之外，TRRS 还需要对数据库表项、数据库集成和适应因特网环境下的安全性等功能提供支持。其他的开发挑战包括提供体系结构的设计工具以及利用 TRRS 数据进行管理等，例如向软件开发者报告相关的 TRRS 产品表项。这些为软件体系结构引入了一个新的动态层面。

14.5　实现模型

最终用户和架构师应在一起审查并贯穿于用例（业务场景、质量场景、易变场景）始终来证实需求的有效。通常这个交流会出现新的或者需要修改的需求，对于需求的任何修改都要标注并结合到随后的其他架构活动中去。通过模型，管理层能够看到可视化

的进展。

大多数系统可以采用快速原型技术生成模型。快速原型技术有利于快速获取产品设计的反馈信息，并对产品设计的可行性做出准确的评估、论证。

14.6 架构原型

在完成上述任务之后，从构建的草图进而发展成产品原型。架构原型是很好的需求验证工具，它能够帮助利益相关人检测系统锲合用户操作的程度。可以使用各种各样的办法构建架构原型，而非编码一种。例如，可以使用故事板来可视化地展现用户使用产品的过程，也可以使用原型工具来模拟过程，以此说明产品是如何运行的。架构原型只是快速构建，作为改进设计的手段，如果在构建架构原型过程中使用了编码，也要尽量避免在最终产品中使用这些代码。

架构框架（Framework）是对系统架构的一种可运行验证工具，通过对系统的 API 定义的编译以及编写小程序来模拟运行的系统。架构框架用于正式计算和工程体系架构，这包括穿越分布式边界的控制和定时。

使用 CORBA 技术，一个架构的规范能够被自动地编译成带有分布式 stub 和框架程序的一系列程序的头文件。通过在框架程序中插入虚拟代码来模拟处理过程，编写简单的客户程序用虚拟的数据来穿越边界发送请求。一些关键的，比如说：高风险的用例被替换的客户程序所模拟。原型的执行被计时以确保与工程约束相一致。

下面是一些架构师可以在架构原型中寻求解答的具体问题。

（1）主要组件的责任是否得到了良好定义？是否适当？

（2）主要组件间的协作是否得到了良好定义？

（3）耦合是否得以最小化？

（4）我们能否确定重用的潜在来源？

（5）接口定义和各项约束是否可接受？

（6）每个模块在执行过程中是否能访问到其所需的数据？是否能在需要时进行访问？

为了构建实际的系统，初始的架构原型需要进行演化。较好的情况是在经过 2 次或 3 次迭代之后，架构变得稳定。主要的抽象对象都已被找到；子系统和过程都已经完成；所有的接口都已经明确定义。

在系统架构开发过程中，利用架构原型，至少有下面的几个好处。

（1）在架构落实之前，让团队成员能自由发表他们自己的看法，并进行讨论，提出建议，对在架构原型中存在的问题进行及时改正。

（2）可以在系统的整体性能上，把握得更好。统一团队成员之间的思想看法和提高系统开发的成功率。

（3）它对系统内部的结构分析与设计也有帮助。

14.7　项目规划

无论什么项目，其最终目标都是要按期、按预算开发出满足用户需求的、高可靠、高性能的产品。在实现这个目标的过程中，项目规划起着至关重要的作用。项目规划是一份已通过批准的正式文档，它根据项目的目标，对项目实施进行的各项活动作出规定，以它为基准跟踪和控制项目，确定未来的行动方案和资源分配，引导项目的实施。项目规划的主要作用是将制定规划的假设和决定以及批准的范围、成本、进度的基线等用正式的文档记录保存。规划的复杂性取决于项目的复杂性，它体现了对客户需求的理解，便于高层管理、项目经理、项目组成员及项目相关人等之间进行交流沟通。

项目规划是基于当前已有的信息，包括过去的经验，当前的目标、范围、组织结构、资源等，工作活动、里程碑、质量目标和风险管理等，其中估算是项目规划的核心。随着项目的进展，信息的增多和理解的深入，估算会不断校正并逐渐地接近实际。项目计划是在规划基础上建立的一组实现任务的活动表，如进度计划、质量活动计划和配置管理计划等。项目管理者通过计划与规划的差异，不断优化和更新计划策略，使项目按规划的要求得以实现，计划的变更是可管理和可受控的。

项目规划是项目工作的纲领，要以此去指导项目的技术和管理活动。项目规划包括如下内容。

（1）项目的目的、范围、目标和对象。

（2）软件生存周期的选择。

（3）精选的供开发和维护软件用的规程、方法和标准。

（4）待开发的软件工作产品。

（5）软件工作产品的规模估计、软件项目的工作量和成本的估计。

（6）关键计算机资源的估计；项目的里程碑。

（7）风险的识别和评估。

（8）工程设施和支持工具计划。

软件项目计划的目标有：软件估计被文档化，以供跟踪软件项目使用。软件项目的活动和约定是有计划的，并形成文档，受影响的组和个人认同与软件项目规划的约定。

14.8　并行开发

14.8.1　软件并行开发的内容及意义

并行开发的意义在于提高软件生产率和改善软件质量。软件并行开发有效地组织可以重复的资源，并附加额外的控制管理技术，使软件开发尽量并行进行，从而达到加快软件开发速度、提高软件生产率、缩短软件开发周期的目的。同时，软件并行开发通过

改善软件过程，达到提高软件质量的目的。软件并行开发以提高软件生产率为目的，对实现软件并行开发的各个方面做了必要的分析，并且给出了可行的解决方案，直接面对软件工程的实施，因此具有重要的应用价值。

软件并行开发研究的内容主要如下。

（1）软件过程及其模型。

（2）并行成分划分。

（3）并行控制。

（4）支持环境。

（5）交互机制与集成技术。

14.8.2 并行开发的过程

要讨论软件并行开发的软件生存周期模型，需要把视野集中到软件开发过程中。把软件系统的开发过程划分为若干个可以并行的成分，这个成分称之为子开发过程。子开发过程是一个动态概念，和操作系统中的进程概念有类似之处。子开发过程可以定义为：子开发过程＝开发小组＋软件对象＋对软件对象的开发活动。或者说，子开发过程是一个开发小组对一个相对独立的软件对象的动态开发过程。

在此，我们把整个并行开发活动看作是一个并行系统，称为并行开发系统。子开发过程是对并行开发系统的一种动态描述，此系统中的实体是开发小组，实体属性是被开发的软件对象，行为是开发软件对象的活动。每个子开发过程完成一个子系统或一个模块的开发任务，当各个子开发过程都完成之后，进行系统集成和测试，最终完成整个系统的开发，如图 14-7 所示。

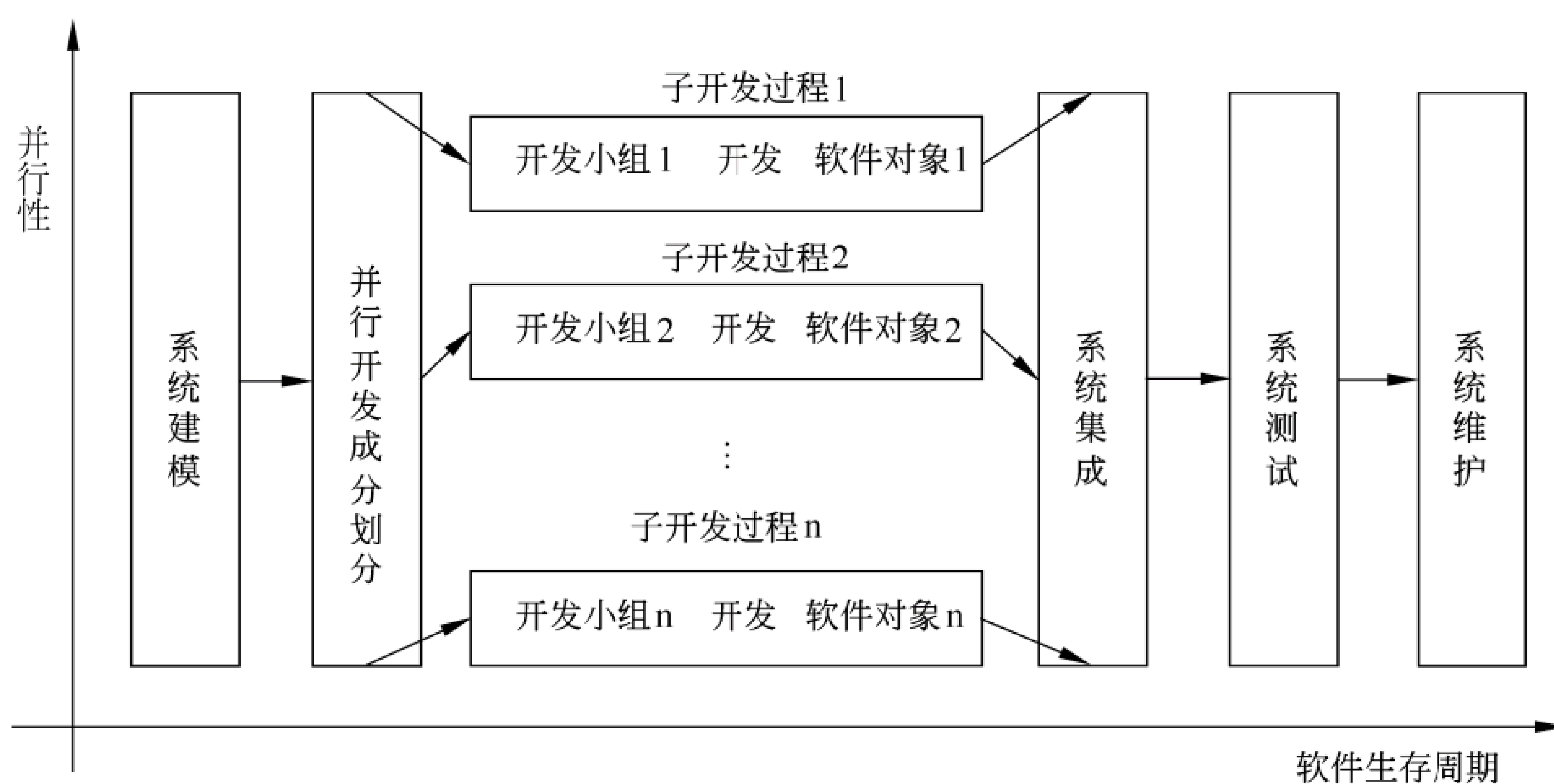


图 14-7 并行开发中的生命周期模型

并行模块的划分是并行开发中的核心问题，模块独立性是衡量软件设计质量的关键。根据并行开发的特征，一个开发小组负责一个模块的开发，如果各模块之间的耦合度低，那么各并行开发过程之间交互作用将减少，为并行开发控制带来方便。有如下两种系统划分的方法。

（1）基于 Petri 网系统模型的动态划分方法。

（2）基于脚本的系统划分方法。

在软件并行开发中，软件过程并行控制（以下简称并行控制）是一个非常重要的问题。所谓并行控制，就是要用正确的方式调度并行操作，避免造成不一致性，使一个操作的执行不受其他操作的干扰。为保证开发出的系统内部各成分间的一致性、相容性，保证系统的正确性和可靠性，就要进行并行控制。通常的并行控制手段有加锁、时间戳、管程、Petri 网和 PV 操作等手段。并行控制模型描述被控制对象的并行行为以及它们之间的关系，是并行控制的依据。

当各个产品开发过程分别完成后，应通过集成技术，把各子开发过程所开发的软件对象集成起来，作为一个统一的应用系统。在软件并行开发的软件生存周期模型中，系统集成和系统测试被分为两个阶段，如果不考虑硬件或系统软件的集成，两个阶段并没有明显的界限。所以，就应用软件系统而言，软件集成的主要问题是集成测试技术。通过集成测试技术，在现实可行的时间内，运用工具尽量去发现尽可能多的软件错误，以保证软件的质量。

14.9　系统转换

系统转换是指运用某一种方式由新的系统代替旧的系统的过程，也就是系统设备、系统数据和人员等方面的转换。

14.9.1　系统转换的准备

在系统转换前，必须认真做好系统设备、数据、人员以及有关文件（如程序说明书、系统操作说明书等）的准备。

除此之外，还需要系统试运行这项准备工作。系统试运行是指在系统没有正式转换之前，选择一些子项目进行的实验运行。需要注意如下两方面的问题。

（1）系统试运行工作的代表性。指在系统试运行工作中所选择的子功能和数据应该尽量接近实际系统运行的需要。

（2）系统试运行中错误的修正。系统试运行过程中用户发现的一些问题，对待这些问题应该以系统分析中确定的系统目标为标准，认真分析产生问题的原因和类型，决定对系统的问题是否修订和如何进行修订。

14.9.2 系统转换的方式

系统转换可分为直接转换、平行转换、分段转换和分批转换。

（1）直接转换。直接转换是当新系统安装完毕能够进行工作后，立即停止旧系统的运行，让新系统投入运行的转换方式。

（2）平行转换。平行转换是新旧系统共同工作一段时间，当证实新系统有较高的可靠性后，再停止旧系统工作的转换方式。

（3）分段转换。分段转换时一次只用新系统的部分功能去替换旧系统的相应部分，逐步完成新系统替换旧系统的转换方式。

（4）分批转换。分批转换是把新系统在小范围内使用，然后再全部推广的转换方式。

以上几种系统转换方式各有各的特点，应根据系统规模的大小、难易和复杂的程度以及企业的具体情况决定系统转换时采用哪种方式。

14.9.3 系统转换的注意事项

在系统的转换过程中，无论采取哪种转换方式，都要注意以下问题。

（1）新系统的运行需要大量的基础数据，这些数据的整理与录入工作量很大，应及早准备，尽快完成。

（2）系统的转换不仅仅是机器的转换、程序的转换，更难的是人员的转换，应提前做好人员的培训工作。

（3）系统运行时会出现一些局部性的问题，这是正常现象。系统工作人员对此应有足够 的准备，并做好记录。系统只出现局部性问题，说明系统是成功的；反之，如果出现致命问题，说明系统设计质量不好，整个系统甚至要重新设计。

14.10 操作与维护

14.10.1 操作与维护的内容

一个系统交付使用后，系统的开发就结束了，系统转入正常的运行操作时期。从系统的生命周期看，只有系统投入正常的操作和维护后，才真正实现了系统。因此，可以说操作维护是系统过程的后阶段。

系统操作与维护的内容有数据管理与维护，包括数据收集、数据整理、数据录入以及数据的分发、数据库管理工作；机器设备的管理与维护，包括硬件维护、机器日常行政管理、系统操作记录和用户服务等；系统软件的管理与维护工作，应用软件的管理与维护工作，代码维护。

14.10.2　系统维护与架构

系统架构的好坏，可维护性是一个重要方面，维护人员应参与架构的评审。系统的可维护性可以定性地定义为：维护人员理解、改正、改动和改进这个软件的难易程度，提高可维护性时开发管理系统所有步骤的关键目的。系统能否被很好地维护，可用系统的可维护性这一指标来衡量。系统的可维护性有如下几个评价指标。

- 可理解性
- 可测试性
- 可修改性

依据信息系统需要维护的原因不同，系统维护工作可以分为以下 4 种类型。

- 更正性维护
- 适应性维护
- 完善性维护
- 预防性维护

某个维护目标确定以后，维护人员必须先理解要维护的系统，然后建立一个维护方案。由于程序的修改涉及面较广，某处修改很可能会影响其他模块程序，所以建立维护方案后要加以考虑的重要问题是修改的影响范围和波及面的大小。然后按预定维护方案修改程序，若测试发现重大问题，则要重复上述步骤。若通过，则修改相应文档并交付使用，结束本次维护工作。必须强调的是，维护是对整个系统而言的。因此，除了修改程序、数据和代码等部分以外，必须同时修改涉及的所有文档。系统维护的步骤如图 14-8 所示。

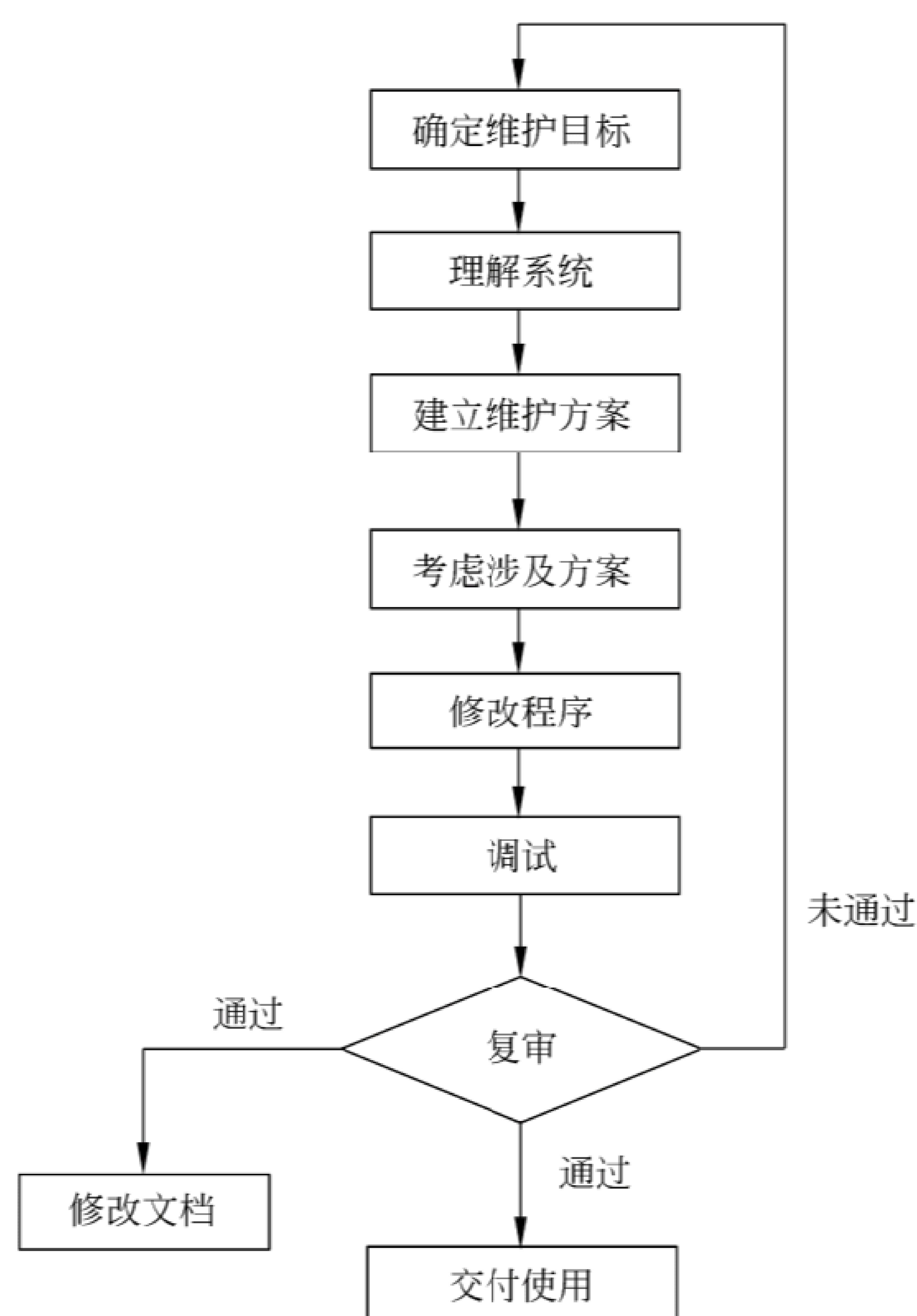


图 14-8　系统维护步骤

14.11　系统移植

14.11.1　系统移植的形式

系统移植的方法有三种：第一种是不修改已有的软件，可以使用的方法有高位互换、仿真功能和虚拟机（Virtual Machine）功能；第二种是修改软件，就是把已有软件资源，

即程序、数据、计算机应用方法及各种说明书转换为与新机器具有匹配性的软件；第三种是重编软件，有从逻辑设计开始、从程序设计开始和从编程开始三种开发方式。

14.11.2 系统移植的工作阶段划分

移植工作大体上分为计划阶段、准备阶段、转换阶段、测试阶段和验证阶段。为了有效地进行系统移植，就得使系统移植工作标准化；配备软件工具实现自动化；还要简化各阶段的工作。下面简要介绍一下系统移植的各阶段工作。

（1）计划阶段。在计划阶段，要进行现有系统的调查整理，从移植技术、系统内容（是否进行系统提炼等）和系统运行三个方面，探讨如何转换成新系统，决定移植方法，确立移植工作体制及移植日程。

（2）准备阶段。在准备阶段要进行移植方面的研究，准备转换所需的资料。该阶段的作业质量将对以后的生产效率产生很大的影响。

（3）转换阶段。这一阶段是将程序设计和数据转换成新机器能根据需要工作的阶段。提高转换工作的精度，减轻下一阶段的测试负担是提高移植工作效率的基本内容。

（4）测试阶段。这一阶段是进行程序单元、工作单元测试的阶段。在本阶段要核实程序能否在新系统中准确地工作。所以，当有不能准确工作的程序时，就要回到转换阶段重新工作。

（5）验证阶段。这是测试完的程序使新系统工作，最后核实系统，准备正式运行的阶段。

14.11.3 系统移植工具

数据不能互换的系统移植时，完整的数据转换工具是必需的。主要有以下几种软件工具。

（1）分析工具：是分析现有软件资源，得到探讨移植方法有用信息的工具。

（2）生成工具：是编制作业控制语言、测试数据、转换工作所需文档的工具。

（3）转换工具：包括程序转换、数据转换和作业控制语言转换。

（4）数据应用工具：使用这种工具不用编文件就可以简便地存取磁带上的数据。

（5）测试、验证工具：作为可分类的工具包括静态、动态跟踪。

（6）管理工具：是管理资源及作业的工具。

系统移植工作需要的软件工具有很多种，配备工具最主要的是在决定移植的工作方法之后，配备移植所需的工具并明确工具的界限。即选出移植工作中的作业项目，使项目系列化、标准化。配备、开发移植所需的工具；对于那些用工具转换的项目，采取相应的措施，进行文档化，使任何人都能以相同的顺序开展工作。这样，就不必制作大量的工具，只将有效的工具组合起来，就可以提高效率。

第 15 章　架构师的管理实践

在实践过程中，软件架构的主要障碍往往在于组织方面而非技术。创造切实可行的软件架构需要对技术的深入把握、良好的认知能力和沟通技巧以及大量艰苦的工作。技术上出色的架构往往由于没有全面地处理好组织管理因素而失败。架构师利用自己的知识影响团队，常被大家认为是无冕之王，因此架构师需要管理技巧。本章介绍了架构师的 VRAPS 实践。

15.1　VRAPS 组织管理原则

VRAPS 是为实践软件架构的组织管理原则提出的，包括构想、节奏、预见、协作和简化 5 个相关联的原则。每项原则都是实际可操作的，原则的提出都来源于构建软件架构的直接经验，并且都可以用来解释实践。VRAPS 模型的焦点在开发和使用软件架构过程的组织管理方面，其应用环境不仅包括建立和部署架构的团队，还包括利用架构开发和利用产品线的团队和使用这些产品的客户。

受益人是指建立并长期保持架构的价值有重要影响的人或组织。受益人一般包括发起人、应用开发人员和应用客户，还可能包括其他重要的参与者，如技术供应方。

（1）构想原则：说明了如何向架构的受益人描述一幅一致的、有约束力和灵活的未来图景。

（2）节奏原则：刻画了一种在整个组织范围内的协调程度，即定期地根据可预测的速度、内容和质量对制品生产进行检查与规划。

（3）预见原则：要在预测未来与检查并适应现状之间做出平衡。

（4）协作原则：解决了如何识别对架构成功关键的团体，以及如何确保这些合作伙伴的有效支持。

（5）简化原则：要求理解组织的结构，了解架构最小的基本特征并最小化架构。

各个原则之间不是相互孤立的，图 15-1 解释了构想原则如何与其他原则交互。构想原则确立了总体方向，使得节奏原则所要求的协调工作能够进行。而一个好的节奏又可以使组织朝着构想原则制定的目标不断提供递增的进展。构想原则中的假设根据预见原则进行测试和验证。在架构演化中，应注意环境的变化，并把这些变化加入到构想中。构想帮助建立准则，以挑选合作伙伴和理解他人给架构带来的价值。这些合作伙伴的约束是一个好的构想的关键要素。构想对简化原则也起到了作用。预期的价值经过解释被运用到架构的决策中，而反过来又帮助完善构想。

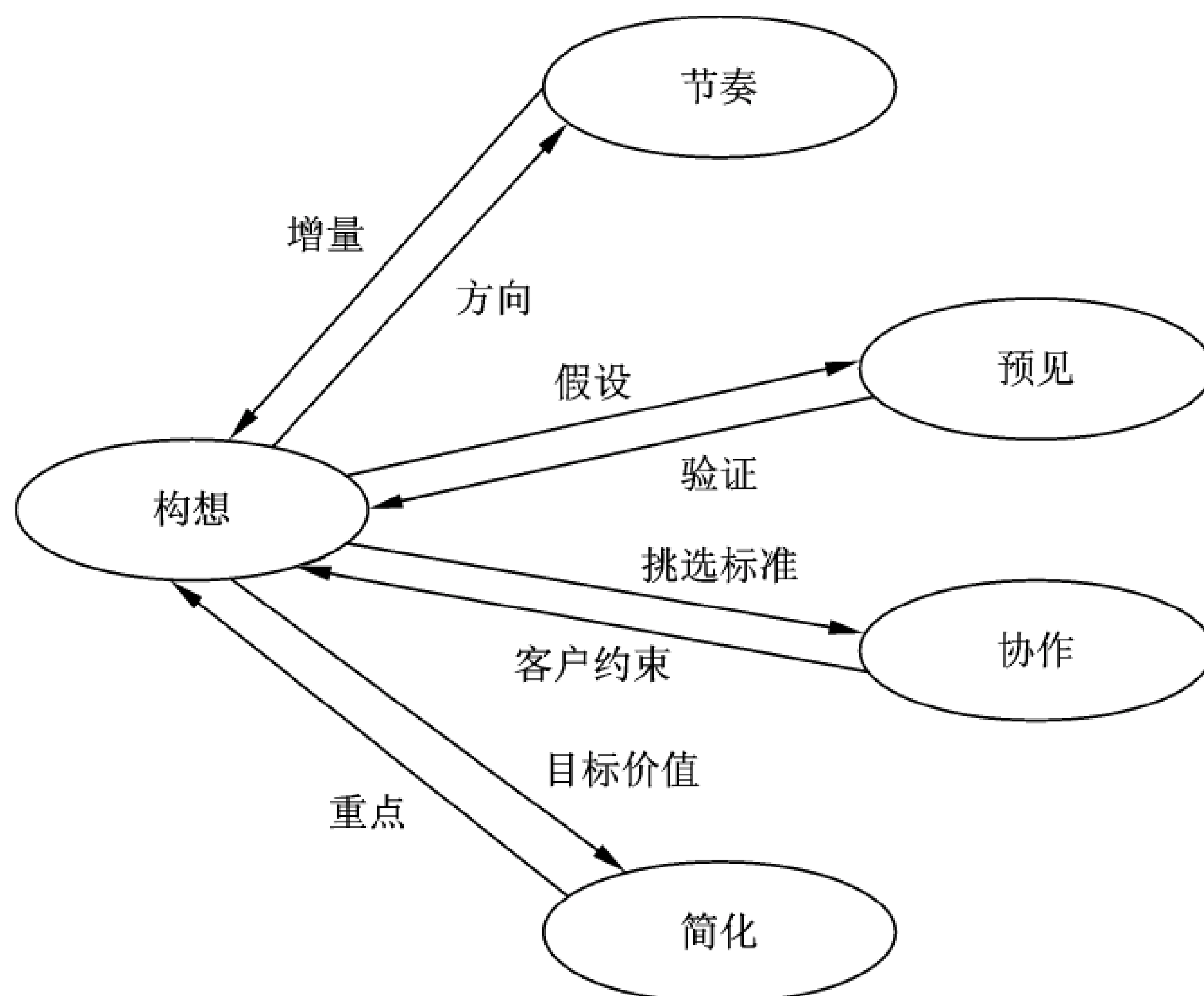


图 15-1 构想与其他原则之间的交互关系

所有其他的原则也是彼此之间相互影响的。例如，节奏原则中的协调组活动如果没有了协作是不可能完成的。通过在每个开发周期中关注最小的关键需求，节奏原则又帮助了简化原则的进行。

15.2 概念框架

为了更好地使用 VRAPS 原则，我们用准则、模式和反模式来对各项原则进行补充。准则用于判断每项原则的实施效果如何。模式描述了开发和使用软件架构时可能遇到的常见问题和解决方法，能够帮助组织改进原则。反模式则描述了组织在实践中可能遇到的陷阱。

1．准则

为了把原则运用到实践中，需要可操作的实施细节。准则把广泛的原则翻译成是否和如何执行原则的细节。

2．模式

第一项原则都附有一组模式，它描述了开发或者使用软件架构时可能遇到的常见问题的解决方法。模式更注重于解决特定情况下的问题，传达了在给定背景和多方竞争因素下针对常见问题的解决方案。

3．反模式

反模式描述了组织在实践中可能遇到的陷阱，描述了不该做的事情，或者用在错误背景下的解决方案，可以帮助更深入地理解原则。

15.3　形成并统一构想

构想描述了架构的未来，提供了架构使用的环境和动机。构想是未来价值到架构约束的映射，构想要成功，则必须把它所能提供的价值与客户的约束相对应。构想也必须是明晰的、有约束力的、一致的和灵活的，从而能够被其受益人理解并有效地运用。

例如，在大型组织中，管理层可能把项目架构师与维护产品构想的高级经理分隔开来。这种距离会维持构想一致性，导致架构难以满足维护要求，引发后期运行成本问题。为了应对类似组织结构产生的复杂性，高级经理和架构师之间建立稳固的、积极的关系以及共享统一的构想至关重要。

把价值映射为架构约束，要求开发人员把约束诸如接口、开发语言和模块边界等映射到特定的客户价值上。促使受益人把约束与客户价值捆绑需要高超的技艺，用例建模是把架构的预期使用与能够被满足的切实的用户目标连接起来的一种方法。例如，识别并表达出似乎无关的用例之间的实质性联系是建立构想的一个重要内容。

15.3.1　形成构想

构想需要维持一致性与协调性。一致性是指受益人的各种期望之间妥协，以及它们与现在和将来的架构之间的需求满足程度。灵活性是指受益人在不破坏架构的情况下，在现有架构之上完成事先没有预料到的需求的容易程度。

一致性并不意味着所有受益人之间拥有一张完全一致的构想视图，而是指各受益人共享的视图根据他们不同的视角保持一致。RUP 的“4+1 架构视图”体现了获得这种一致性的方法。RUP 通过逻辑视图（Logic View）、实现视图（Implementation View）、进程视图（Process View）、部署视图（Deployment View）和用例视图（Use case View）建立了架构视图。这些架构视图的不同点在于，它们根据不同目的表示系统（例如，用例视图表示了系统的最终用户功能）。

架构师常负有将现实引入业务构想和将构想变成现实的责任。架构师可以推荐技术，包括如何以及何时采用这些技术，由此来帮助确定业务构想的哪些部分可以在短期内实现，以及各部分实现的次序。

架构师更像管理者而并非实施者，Dean Thompson 说：“作为架构师更多地意味着权衡业务、组织运作和使用技术，而不仅仅是技术细节。”例如，架构师应该全面研究整个组织，找出各利益方关注的重点，然后妥善平衡，建立符合主要关注问题的架构描述。

多方整合能促进构想的形成。多方整合是组织各利益方的机制，用于确保获得构想并使其稳定；是指在一个公共的组织层次上对信息、决策和资源进行协调。多方整合能使从事硬件设计的基层经理理解软件设计和开发人员，以及市场营销、客户支持和市场营销的同事的期望，包括增强组织与客户和外部供应者沟通的能力。

Thompson 归纳了形成架构构想的三步方法：清楚明确地阐述一条迫切的客户价值；将客户价值映射为少数特定的能解决的问题；将以上问题转译成一组特定的约束条件。

成功的架构师用明确的客户价值映射规划未来，以使用户及它们的客户能将其与约束联系起来。架构师必须格外关注产品开发人员和最终的客户，而且为了成功，还要发动所有其他人做类似的事情。

15.3.2 将构想原则付诸实践

下面的准则、反模式和模式能帮助建立、形成、维护一个被共享的构想，将构想原则付诸实践。

用于检验构想原则是否起作用的准则如下。

（1）架构师的构想与发起人、用户、最终客户期望实现的目标是否保持一致。

（2）实施人员是否信任并使用架构。

（3）关于架构和构件的潜藏知识对其用户（开发团队）是否是可见的、可获得的。

构想原则中准则到模式、反模式的映射如表 15-1 所示。

表 15-1 准则到模式、反模式的映射

准则——如何度量	反模式——不该做的	模式——可以做的
架构师的构想与发起人、用户和最终客户期望实现的目标是否保持一致	风险后置	前后一致
实施人员是否信任并使用架构	墙头草	三个臭皮匠
关于架构和构件的潜藏知识对其用户是否是可见的、可获得的	一叶障目	轮转工作

下面详细介绍与各准则相关的反模式和模式。

准则 1：架构师的构想与发起人、用户、最终客户期望实现的目标是否保持一致。

为了获得一致、迫切和灵活的架构，需要产品线经理、架构师和实施经理等达成共识。而如果没有阐明用户价值，则会导致构想脱离了重点。

与准则 1 相关的反模式与模式如下。

1）反模式：风险后置

形容这样统一受益人的构想：用最小的妥协、最大的优化规划出一个构件以满足所有冲突利益的需要。这种统一方法的问题是，设计出来的构件往往在理论上可行，但实际运行中出现风险。

一条新产品线的架构师或经理，需要开发看似很棒但实现有风险的构件。这些要完成构件可能需要打破“物理定律”才能完成，这些风险可能在制品交付的最后才能显现。可是有一批工程师仍然坚持开发这些构件。风险构件被安排到最后完成，以为这样可以有时间消除风险。可是，当计划好的完成日期临近时，依然无法交付。尽管架构在演示

的幻灯片上运转良好，实际上却无法正常工作。

面对这种情况，需要分析并阐明风险，向高级经理提供一个选择，要么承认风险，要么调整任务。

2）模式：前后一致

要求推动架构投资的高级经理积极地维护构想，并防止构想受到短期压力的影响。

一个公共的架构被几个产品共享，它已经变得比预期要复杂得多。而客户们针对每件产品又提出了以前没有预计到的功能特性。如果加入这些功能特性，则不能保证进度，但如果不开发这个特性则可能失去一位重要客户。

这样的情况下，需要评估架构构想的质量和稳定性。只有当两者都正常时，才能采取进一步行动。如果该新特性不属于原来的产品构想支持的代价范围，那就应该放弃开发这个新特性。如果构想不明确，在短期内就交付很可能导致大量缺陷。此时应与客户、架构师和销售、产品、支持人员以及开发经理一起加强产品的构想。如果这个特性确实属于一个稳定的产品构想，那么应该在开发组织内核实这种一致性。

准则 2：实施人员信任并使用架构。

只有使用架构，才能从中获得价值。然而，要让开发人员对架构构想充分信任需要做更多的工作，仅仅靠不断兑现承诺是不够的。开发人员需要把他们对构想的了解与他们认为对下一步行动有用的东西联系起来。开发人员在利用架构的过程中，可能会使架构向许多不同的方向发展。一个良好架构应与构想保持一致，同时又能满足用户的需求。

与准则 2 相关的反模式和模式如下。

1）反模式：墙头草

描述了这样一种情况：因为没有良好的构想，导致架构方向在竞争和客户压力的影响下经常改变。这种构想永远不能达到稳定以便有效地被共享。

在开发过程中，经常会出现这样一种情况：来自客户、竞争对手和高级经理的压力使得需要在一次发布的中途加入一些代价高昂的功能，高级经理甚至可能在没有咨询架构师的情况下决定提供这些功能。可是一旦交付了某个激进的功能，组织马上就会陷入对该功能的支持工作中。以后的发布会因需要提供向后兼容而变得更为复杂。

在这种情况下，要做的是理解并阐明构想。方便变更需求是成功构想的一部分，要对其进行规划。高级主管要与架构师一起紧密地工作，理解变更的后果并做出正确的权衡。在建立了构想之后，可以用前后一致模式来评估特定的变更建议。在把功能特性加入到发布之前，要坚持达成一致意见。这需要一种固定的机制，以使达成一致成为正常业务的一部分。如果架构满足了高层的约束条件，那么在细节的实施方面可以允许更多的灵活性。在最极端的情况下，解决墙头草问题可能需要寻找新的、期望更为接近的受益人。

2）模式：三个臭皮匠

三个臭皮匠：反映了这样一种认识，即架构师并非总是架构构想的来源，架构师和客

户一起充实、完善构想。

一个共同的架构或平台是产品线战略的关键，架构发起人希望产品团体能开发出“杀手解决方案”，而同时又能避免追随新的标准、潮流带来的困扰。

在这种情况下，需要抵制创建一个无所不能的架构的诱惑，建立一种能让架构师及其用户都能丰富、实现功能特性的构想。高级经理只提供构想、目标和原则，把架构和平台留给架构师，把实现细节留给合适的团队或层次。一个成功的产品线架构必须能为适应市场变化，能适应和采用新技术，能解决在概念阶段还不知道的但变化场景可预见的问题。

这是一项很少有架构师能独立完成的任务，然而，通过高级经理建立正确的顶层业务构想，架构师采取实现构想的正确行动就能取得成功。

准则 3：关于架构和构件的潜藏知识对其用户是可见的、可获得的。

1）反模式：一叶障目

一叶障目：发生在这种情况下：开发人员过分专注于应用，以致不知道其他架构解决同类问题的通用的解决方案。

在一个组织中，工程团队正在实践代码所有制模式，工程师们都把精力集中在自己眼前的任务和职责上，没有把自己当作一个共享资源的看护者。陷入本反模式的工程师的视野很窄，面对其他开发同事的请求往往只求解决问题，不愿意多加考虑。这样导致的工作结果可能变得非常复杂、脆弱而且容易出错，这是由一名工程师所无法预见的情况造成的。

要解决上面的问题，需创造一种分享知识的愿望。例如，工程师培训等方式就可以起到一些作用，整个组织的人员启用知识管理平台也能促进产生这种愿望。

2）模式：轮流工作

轮流工作：要求参与架构的工作人员轮流在架构的不同部分上工作。这样能使他们对架构有更多的了解，并有机会发展非正式的人际网络。

当一个产品线的销售增长到一定程度，一个单一的、聚集的团队已无法支撑架构和实现，人员被重新组织到地理上分布不同的团队中。已有的沟通管道与新的形势已经不相适应。

通过帮教制（Apprenticeship）模式阶段性地轮流交换构件的所有权可以解决上面的问题。组织和鼓励构件的前任负责人抽出时间帮助新的负责人，轮转周期应该尽可能地与发布进度保持同步。该方法能使开发人员更容易地发现如何找到有关构件的知识，团队成员掌握某个构件或者在出现问题时知道应该问谁的可能性也大大增加了。因此，意外发生的次数减少了。

15.4 节奏：保证节拍、过程和进展

节奏原则使得软件架构在跨越组织边界的情况下开发和使用成为可能。由于许多参

与开发和使用架构的团体是自治性的，不可能自上而下地协调这些团体，节奏原则提供了一个随时间变化的框架，可使团体同步各自的活动与期望。有了节奏，参与者就能知道何时关心和应该关心哪些活动。不仅计划中的活动可以被协调管理，节奏原则也可以协调那些非正式的但很关键的活动，例如团体间的交流，这样参与者就可以知道何时应该或不应该提出对于信息或支持的要求。

15.4.1　节奏定义

节奏是一个架构团体内部及它与客户和供应者之间反复出现的、可预测的工件交换活动。节奏有三个元素：速度、内容和质量。速度是指一个团体与另一个团体之间同类型交接发生的频率，例如架构团队与产品开发工程师之间。如果交接的时间是可预测的，移交则容易管理。稳定的发布计划是速度的一个例子。内容是指一个团体向另一个团体提供的价值。例如，一个团体开发一种新的或者要修改的特性被另一个团体用于满足某种需要。质量的含义是遵循开发过程确保架构没有缺陷。组织可以通过省略非增值的步骤来加快速度，但是如果重要的流程被截掉了，节奏就会遭到破坏。

节奏在团体和组织之间与内部提供一种协调活动的稳定力量，帮助移交管理。当节奏很强时，受益人能培养很强的预见、实施移交和交接的技能。节奏还能驱动活动完结，拥有良好节奏的组织通过建立有规律的阶段间隙来推动评估、再评估和其他工作的进展。

15.4.2　将节奏原则付诸实践

没有建立节奏会导致客户不满意、不期望的错误发生和工件无法一起工作。只有当以下准则出现时，才说明节奏原则起了作用。

（1）经理们定期地再评估、同步和调整架构。

（2）架构用户对架构发布的进度和内容具有高度的信心。

（3）通过节奏协调明确的活动。

节奏原则中，准则到模式、反模式的映射如表 15-2 所示。

表 15-2　准则到模式、反模式的映射

准则—如何度量	反模式—不该做的	模式—可以做的
经理们定期地再评估、同步和调整架构	一步成功	发布委员会
架构用户对架构发布的进度和内容具有高度的信心	超敏捷	舍兵保帅
通过节奏协调明确的活动	销售未检验的产品	同步发布

准则 1：经理们定期地再评估、同步和调整架构。

好的节奏需要有规律的节拍。对于架构组织的管理者，这意味着他们必须在稳定的间隙上再评估、同步和调整他们的架构计划。节奏的节拍还能提供一个计划过程的框架。具有良好节奏的一个组织用节拍而不是时间来衡量进度。

相关的反模式和模式如下。

1）反模式：一步成功

一步成功：是指当组织变得过于专注地向市场推出某项功能特性而导致内部节奏遭到破坏时发生的情形。组织被竞争所蒙蔽，全身心地专注于向市场提供该特性，却削减了质量，甚至可能破坏本来的节奏。

相关的解决办法有：把关键的功能特性作为团队节奏的一个组成部分来实现。围绕一个特定的主题来进行一次特定的发布，利用主题帮助抓住市场上的机会。如果关键特性特别复杂，则采用几次迭代来实现。但如果在实现关键特性的时候难以保持节奏，说明该特性的风险和复杂度比预计的要大，需要重新规划。

2）模式：发布委员会

发布委员会：描述了一种协调参与发布新架构的相关各方的方法。该模式向经理们介绍了一种在架构发布的最后阶段再评估、同步和调整架构的方法。

定期举办由组织中每个关键受益人参加的正式会议以引导发布的进程。在会议中，要复审产品功能特性和优先级的变更，从而使产品文档、市场承诺、公共关系、测试和开发保持一致。在适合的地方采用测量指标度量发布的进展，分享责任和依赖，做出如何前进的决定，并记录和传达会议的决定。参加委员会的成员应该保持稳定，会议成员的组成应保持一致，与会人员也应该有足够的决定权。

准则 2：架构用户对架构发布的进度和内容具有高度的信心。

如果架构用户不信任架构发布的进度和内容，那么用户就可能不采用新的架构发布，或者可能选择另一个架构。用户对架构发布的进度和内容缺乏信心是一个警告信号，说明没有建立一个良好的节奏。

与准则 2 相关的反模式和模式如下。

1）反模式：超敏捷

超敏捷：发生在组织试图在开发过程中抄近路以维持稳定的发布节拍的时候。该反模式对用户所期望的架构质量和内容进行了妥协。

过程执行的合适方法取决于组织文化。在有些组织中，阶段性的软件审计可以用于保证过程被遵守。然而，在许多组织中，审计并不是一种改变或约束行为的有效方法，高层管理的行动能更直接地改变组织行为。是否已经分配了足够的资源来执行计划中的步骤，经理们是否创建明确的目标来保证对节奏的维持都有非常重要的影响。

2）模式：舍兵保帅

舍兵保帅：探讨了组织如何通过把不太重要的特性移到后面的发布周期以保持一个节拍。通过保持节奏，该模式可以使用户获得对架构发布进度更多的信心。

如果对构件的修订看上去无法及时完成时，而且该修订并不非常重要，应该尽快向受益人说明。在不对延误构件做变更的情况下，继续发布架构。为了避免因用户没有阅读或看见特性变更说明而造成的问题，应该确保从预发布开始就放弃该特性，这样用户

就能在 α 或 β 测试中体验到变化，而不是在正式的产品发布中。可以在以后的发布中再把该构件的变更加进来。通过上面的方法可以保持发布的速度，使架构发布后的活动计划能按进度进行。该方法还能推动构件负责人按时完成他们的修订工作。因为架构的发布实现了承诺，开发人员增强了信任感，他们对下一次如期发布也有了更多的信心。

准则 3：通过节奏协调明确的活动。

软件架构的受益人分布在许多不同的组织中。一个共享的架构节奏能帮助这些自治团体跨越组织边界协同工作，因为它能帮助建立关于关键事件何时发生以及如何发生的共同假定。例如，如果一位产品开发员知道每年有一个架构主发布和若干季节性的维护发布，他就可以根据预期的架构发布安排产品发布时间，更好地利用新的架构特性。

与准则 3 相关的反模式与模式如下。

1）反模式：销售未检验的产品

销售未检验的产品在此情况下发生：一个组织试图实现定期的建立，但这些建立经常编译失败或无法通过自动测试。这表明协作的失效。

由于团队不把编译和测试用例的失败当回事，认为在以后的开发过程中能消除这些不一致的情况，导致积累下来的问题越来越多，无法按时发布。

在这样的情况下，要确保对定期建立的承诺。管理层必须明确无误地告诉开发人员，定期建立应该成功。软件建立不仅应该包括编译产品，还应该包括某种形式的自动测试。对定期建立的流程进行修改，防止在修正失败的建立之前开展新的工作。同样地，以前曾经通过的失败测试用例应该马上处理。

2）模式：同步发布

同步发布是一种把节奏理念扩展到组织边界以外的技术。该模式提供了一种同步架构团队及其用户的活动的方法。

和你的合作伙伴一起确定交付架构特性的先后顺序，以便他们利用架构开发产品。应尽可能在架构的早期发布中包括这些特性。如果架构中的一些变化需要互补产品做出重大变更，那么应让这些变化出现在最早的预发布中。应该告知合作伙伴何时能够获得哪些特性。作为调整早期发布方式的回报，应与合作伙伴签订协议让他们把包含或需要你的架构的产品迅速推向市场。

15.5 预测、验证和调整

为了使对软件产品线的长期投资能产生回报，组织必须确保架构满足许多应用的需求。组织应能够预见变化并对变化做出反应，包括那些在设计架构时还没想到的需求。架构必须能够适应新的技术、标准、市场和竞争对手。刚开始设计架构时正确的假设几年以后可能就失效了，这就要求组织必须能够对架构进行预测和演化。

15.5.1 预测、验证和调整的定义

预见是指建立和实现架构的人员根据变化的技术、竞争和客户需求预测、验证和调整架构的程度。

软件架构师不可能总能预测到未来。但是，既然一个成功的架构将被持续使用很长时间，架构师至少要对未来将发生什么做出合理的猜测。架构师必须考虑架构用户可能怎么变化，竞争形势将如何改变，未来的运行环境是怎样的。架构必须能够适应新的组织结构，特别是在一些像银行业这样合并和接管司空见惯的领域中。许多计划建立在对未来的假定上，但是预测意味着这些假定是作为架构描述的一部分而明确表述的。例如，有关的假定可能基于这样一种未来的情况：处理器速度依然遵循摩尔定律，在以后的 10 年里每 18 个月翻一番。当然，除非架构师有预知未来的超自然能力，否则这些预测不可能总是正确，所以需要验证。

验证不仅局限于传统软件工程的测试和检查技术，也包括对架构的基础假定的测试。例如，用户真的想要计划好的东西吗？现有的技术能实现用户的需求吗？分析这些假定的重要原因是，架构师及其发起人做出了许多关于架构的艰难决策。在架构成型前要对这些假定进行检查和确认，否则会导致代价高昂的错误。

软件架构的长期成功依赖于对假定的变更和通过预测及验证所获信息的适应程度。调整就是对架构计划及架构本身修正以加入新特性，从而能参与新兴市场的竞争或者在新的环境中生存。因此，调整要求组织具有敏捷性。调整可能不仅包含架构本身，还包括计划，甚至整个架构构想。

15.5.2 将预见原则付诸实践：准则、反模式与模式

下面的准则、反模式和模式为帮助判定组织在预见验证、调整架构方面提供了指导。当以下情况发生时，说明预见原则发生了作用。

（1）不断增强架构的响应能力：预见到的风险和架构客户及其客户的需求；市场驱动的标准和演变的技术；战略性业务方向的改变。

（2）通过快速复审和开发周期，评估技术和业务上的风险与机会。

（3）当认识到关键的估计或假设有错时，及时调整功能特性、预算。

表 15-3 准则到模式、反模式的映射

准则——如何度量	反模式——不该做的	模式——可以做的
不断增强架构的能力响应 （1）预见到的风险和架构客户及其客户的需求 （2）市场驱动的标准和演变的技术	遗漏细节	示范区

续表

准则——如何度量	反模式——不该做的	模式——可以做的
战略性业务方向的改变 通过快速复审和开发周期，评估技术和业务上的风险与机会	品尝未熟的果实	架构复审
当认识到关键的估计或假设有错时，及时调整功能特性、预算	创造奇迹	外包

准则 1：不断增强架构的能力以响应预见到的风险和架构客户及其客户的需求，市场驱动的标准和演变的技术，战略性业务方向的改变。

与此相关的反模式与模式如下。

1）反模式：遗漏细节

遗漏细节：描述了发现一个明显的功能特性被遗漏时的尴尬经历。每个人都关注发布的强大新特性，以致忽视了一些用户必不可少的功能。

对这种情况，需要识别关键用户群，和他们一起找出最重要的需求。这意味着要有一位对这些用户群有着深入了解的专题事务专家的参与。要调查的问题包括“哪些产品会建立在这个架构之上？”，“哪些产品是最重要的？”该方法也可以用来指导架构方向的改变。调研的覆盖面很重要，需要让高级经理了解这一过程以使他们在敦促实现其他高级特性和技术的时候，不会在无意中破坏这些基本特性的交付。

2）模式：示范区

示范区模式应用在如何决定哪个产品应该引入一个新架构的情况下。

挑选一个项目初步实现架构。该项目的客户渴望采用新技术，而且也愿意容忍获得该技术时可能存在的不便。当示范区项目投入使用后，架构师将从实际使用中得到关于架构的有价值反馈。在架构大范围应用之前，缺陷将被发现并解决，这也意味着缺陷的修订受后向兼容问题的约束较小。成功的示范区项目可以在建议其他项目时作为参考。

准则 2：通过快速复审和开发周期，评估技术和业务上的风险与机会。

1）反模式：品尝未熟的果实

品尝未熟的果实：说明了当架构师没有考虑其用户的客户，而让其用户支持一种未成熟的技术时发生的情况。

在发起人的要求下，架构师采用了一种新技术来建立下一代架构，团队希望通过它胜过最接近的竞争对手。客户对架构糟糕的性能和各种各样的差错感到很失望，客户并不关心底层技术，他们只需要那些能帮助他们实现目标的东西，一些被以前的换代或升级害苦了的客户则对关于未成熟技术的承诺极度不信任。

在这种情况下，要审慎地选择引入新技术的正确场所。在引入后，要为最初用户提供额外的支持。在选择新架构解决方案时，必须愿意修改技术不完善的部分使其适合一个实际可用的解决方案。不要假定一个未经验证的架构能实现所有的承诺，应该分别在

开发人员和产品用户的特定环境下测试你的解决方案。即便如此，还必须向采用新技术的用户提供大量的支持，要谨慎地设定用户正确的期望，留意他们可能遇到问题的迹象。

2）模式：架构复审

架构复审模式总结了怎样针对开发中的架构组织执行一次有重点的专家评估，以揭示有重大影响的问题和机遇，例如假设的冲突、可重用的现有方案等。

该模式提出在开发周期的关键时刻成立一个架构复审委员会以检查架构。一旦需求基线初步确定就应该进行首次复审。复审委员会的成员应该包括有经验的架构师、架构小组成员，可能还有客户，人员不要太多，最多七、八个人。在早期复审中，应检测各种假设，看看市场上是否有可购买的解决方案，并进行其他条理性检查。后期的复审应验证假设，确认架构是否满足了需求。注意要让这些复审保持重点。

这种模式可以避免增加成本，因为复审能够在开发过程的早期发现缺陷，这样就可以及时修正。复审能发现可以取代新的开发活动的构件。此外，它还增强了客户对架构提供已承诺能力的信心，从而促进客户使用架构。

准则 3：当认识到关键的估计或假设有错时，及时调整功能特性、预算。

1）反模式：创造奇迹

创造奇迹：描绘了当足够的证据显示基础假设和估计已经完全偏离目标时，对架构开发和实现计划不作任何修改将发生的情况。

解决方法分为如下两个部分。

（1）找到架构的基础假设并积极努力测试这些假设。架构复审和示范区模式提供了获取这类信息的手段

（2）一旦发现错误的估计或假设，必须准备好对此采取行动。这可能意味着调整项目进度、功能特性或者启动意外处理计划，此外还包括提醒客户并重新协调进度和发布的内容等。

还应该特别小心那些遏制信息和创意传播、掩盖错误假设的证据的组织文化。无论何种情况，都应该确保把足够的资源编入预算计划，使得当不可避免的意外发生时，有可分配的进度和人员。

2）模式：外包

外包模式展示了怎样适应这种情况，即客户要求的新标准或技术并不属于当前或计划中的核心能力。它提供了指导以说明何时及怎样选择一个已有的第三方构件，或者与供应者合作。

如果存在第三方构件，应考虑采用。如果没有这样的构件，那么组织应该找到合作伙伴来开发和支持该构件。要确定潜在的合作伙伴是否把你需要的构件视为其主营业务的一部分。例如，他们是否能够把它卖给许多其他的客户；评估他们交付和支持该构件所需的特定工作量。把潜在的合作伙伴当作供应商和业务伙伴以评估其能力和信用。基本的规律是，他们必须为你做的专门开发越多，信任程度就要求越高。类似地，信任度

越低，你面临的进度和财务风险就越高。如果发现一个非常可靠的潜在供应商，就应该外包构件开发。

15.6　协作：建立合作型组织

协作也是软件架构成功的关键之一，因为不同团体参与者对架构的开发、实现和使用都是很重要的。这些团体跨越了各种各样的组织边界，如团队、地理位置、部门甚至公司。每一个对架构关键的团体必须知道如何使用、努力改进架构从而为自己的利益服务。协作原则解决了如何识别对架构成功起关键作用的团体，以及如何确保这些合作伙伴的支持等问题。

15.6.1　协作定义

协作是指架构受益人保持明确的、合作的角色并将其所提供和获得的价值最大化的程度。合作是指受益人彼此之间存在一些共享的预期，应该明确表示出达到或未达到预期会有哪些奖励和惩罚。成功协作不仅仅要求架构负责人满足契约条款，合作伙伴还必须采取行动确定和提供预期价值，根据已达成的条款给出特定问题的解决方案。

15.6.2　将协作原则付诸实践：准则、反模式与模式

协作很容易理解，但将其付诸实践并不简单，当许多团体必须在一个组织内（外）的同一层次上进行合作时尤其如此。正式定义的协作网络与非正式协作网络决定了一个软件架构能否成功。以下准则提供了一种方法用来确定受益人为了使架构与产品服务的价值最大化而进行合作的程度。当出现以下几种情况时，说明协作是有效的。

（1）架构师不断地努力了解谁是最关键的受益人，他们如何贡献价值，以及他们需要什么。

（2）受益人之间达成明确和强制性的契约。

（3）通过社会行为制度和非正式规范强化合作。

表 15-4 介绍了准则到模式、反模式的映射。

表 15-4　准则到模式、反模式的映射

准则——如何度量	反模式——不该做的	模式——可以做的
架构师不断地努力了解谁是最关键的受益人，他们如何贡献价值，以及他们需要什么	光说不做	了解你的受益人
受益人之间达成明确和强制性的契约	不记录讨论结果	互惠互利
通过社会行为制度和非正式规范强化合作	非正式时间做正式工作	杜绝意外 和 HR 密切合作

准则 1：架构师不断地努力了解谁是最关键的受益人，他们如何贡献价值，以及他们需要什么。

满足受益人的需求说起来很容易，实施起来要困难得多。挑选一批集中的首要客户，找出保证他们参与需要做些什么，然后交付这些内容，这样做可以增大成功的机会。

与准则 1 相关的反模式和模式如下。

1）反模式：光说不做

光说不做描述了这样一种情况，即架构师知道了用户的需求却遗漏了为了向他们提供有价值的东西所应该做的事情。

架构师忙于其他事务，没有与开发人员进行稳定的交流，各产品团队按照自己的理解开发并升级了产品，放弃了原来同意的清晰的接口。

与许多反模式一样，该反模式中最困难的部分就是，当发生这种情况时如何识别它。当你认为“我们可以以后再补充这些细节”时，应该保证你和你的团队至少理解一些关于开发人员如何从平台获益的明确的例子。模式“了解你的受益人”提供了一种掌握需要哪些措施让受益人参与协作的方法。客户互动模式提供了一些明确、简单和直接的规则与建议，有助于发展有效的协作关系。

2）模式：了解你的受益人

本模式说明了如何利用价值链来识别关键受益人，积极听取他们的意见并获得承诺与支持。

把架构成功的构想与那些最符合合作伙伴能力并积极去做的事情的活动统一起来。阐明设想中架构所能提供的价值，例如架构如何帮助现有产品取得一致的用户界面或者继续保持市场优势，或者是能打开一个针对新产品的全新市场。在初步阐明构想之后，确定潜在的合作伙伴以及他们的能力和利益如何与构想保持一致。

准则 2：受益人之间达成明确和强制性的契约。

1）反模式：不记录讨论结果

不记录讨论结果说明了当一个架构团队回避采取必要的行动与其最直接的用户达成明确的契约时会发生什么情况。当用户们失去兴趣时，虽然对话仍在继续，但是讨论已经失去了实质内容，而且通常会浪费所有人的时间。

要确保取得对关键受益人的利益与职责的明确理解。把这些认识记录下来，当互动变得消极或者缺乏建设性时，可以求助于这些文件。这种做法总是很重要，对于那些对别人有强烈影响的参与者而言尤其关键。当状况似乎要失控时，回到当初的约定可以把架构团队从漩涡中解救出来。

2）模式：互惠互利

本模式介绍了一些非常重要的做法，用来建立足够稳固的关系以保障软件架构的共享和成功使用。

互惠互利要求在合作伙伴之间进行公平、主动的价值交换。当共享一个架构的团体

之间的关系定义好之后，应该对正式和非正式的契约复审以保证公平的交换。预算中应该包括代码负责人响应其他团体请求所花的时间。要对各个团体支持其他团体的程度进行衡量，而不仅仅评估他们完成自身任务的情况。

准则 3：通过社会行为制度和非正式规范强化合作。

协作包括正式和非正式两方面，为了真正巩固协作，需要用社会行为制度和非正式规范来促进合作。

1）反模式：非正式时间做正式工作

非正式时间做正式工作介绍了这样一个情况，即一位工程师申请修改某个构件以便让其他团体使用，却得到一个令人困惑的答复："你可以做，但是要用你个人的时间。"

让工程师利用业余时间修改，架构师就失去了控制其过程和结果的能力，他可能没有采用组织的文档标准，诸如同级复审等步骤甚至连测试也有可能被删减或完全忽略。如果这种产品加入到其他团队的工件中，这位工程师在需要完成日常任务同时，还接到大量要求提供支持的请求，导致工程师精疲力竭。

对于以上情形，要制定计划奖励工程师花在共享构件上的时间，尽早兑现奖励能减少工作量和大量压力。应仔细考虑如何处理将来这一构件成为多个外部项目的关键的可能性，在权衡利弊时必须根据组织纪律和常理判断，包括企业文化、管理的洞察力、进度压力的程度以及当前状况的细节。很多组织把员工用于开发、维护被团体或项目外部所共享的解决方案的时间编入预算，这样能够预防工程师在利用非正式时间做正式工作开发时对项目的代码偷工减料，并确保你的小组对其他团队或项目的支持能力。

2）模式：杜绝意外

该模式描述了如何在不失去依赖你的构件的其他团体信任的情况下，调整对进度或功能特性的承诺。

要尽早提醒用户注意变更，并及时协商解决方案。在决定变更的内容之前，要确保通知、咨询了构件的用户。让他们了解虽然现在的做法对软件架构能产生直接的影响，但其实它们有着更为广泛的应用。

3）模式：和 HR 密切合作

和 HR 密切合作介绍了这样一种做法，即提拔雇员并不仅仅根据个人的技术技能和经验，还要考察其有效地、合乎道德地利用非正式人际网的能力。

软件开发是一种社会活动。可是，很多工程师属于内向型性格，工程师需要与他人交流以获得完成其工作所必需的信息，大部分高级技术岗位要求能迅速获得广泛的潜藏信息。有着广泛非正式人际网的工程师比没有这种网络的工程师能获得质量更好的信息。

在作提拔决定时，要考察一名工程师的非正式人际网的有效性。此时，应找出具体的事例，例如，这位工程师是否通过团队外部的合适人选，获得了曾困扰其同事、阻碍项目进展的问题的答案？如果组织已有晋升的明确标准，那么也可以对此标准做类似的

调整。经理们应该避免破坏非正式人际网。

15.7 简化：澄清与最小化

架构师和高级经理必须协力保持架构和组织的平衡。聚焦于客户和业务价值，为架构师提供了方向和指南。确定关键价值是不容易的，尤其是当新客户和新产品的加入使架构偏离原来的方向时，困难会显著增加。构想定义了这种关键价值，而且为实现价值建立了约束。简化则将构想翻译成产品。

简化软件架构的原则概念上看似简单，而实践中它要求对价值非常坚定地专注，以及对架构所生存的组织的理解和支持。架构师必须了解架构最小的基本特征。简化原则还要求通过努力，把这些特征传达给实现架构团队的每一位成员。

15.7.1 简化定义

简化是指将所作用组织与环境都进行巧妙地理解与最小化，组织形成架构并且思考架构。在决定简化架构时，应当留意组织的结构；否则，你会发现你所做的改变只是暂时的。因此在简化架构之前，必须澄清组织和架构。

澄清组织意味着真实地理解你计划部署架构于其中的组织结构及其影响力（force）。架构对架构团队和客户都必须是清晰的。在简化架构之前，架构师必须精确地知道架构被期望做什么和如何完成这些任务。有时候看似很容易的任务，结果实现起来却很复杂，如果这些复杂性没有被理解清楚，那么建立的架构就可能完全不适合目标任务，而这样的架构会使实现更加复杂。澄清架构就是提供用户所需要的细节。

如果一个组织具备简化、协作和节奏等技能，长期共享架构就能够最小化代码、文档和过程。不必去新发明大量新的代码，却可以开发一种被工程师跨组织共享的公共语言。共享也能促进理解，因为它能最小化用同样术语描述完全不同概念的风险。共享并不能自动产生最小化，在有些不好的组织情况下，共享可能导致架构膨胀。

15.7.2 将简化原则付诸实践：准则、反模式与模式

当以下准则都满足时，说明简化原则起作用了。

（1）开发人员长期使用架构，减少了总成本和复杂性。

（2）架构小组明确理解关键最小需求，并且将其构造成多应用共享的核心元素。

（3）通过长期的预算和行动确保当相关元素没有被共享、增加了不必要的复杂性时，或者是因为有明确的业务理由时，把相关元素从核心移走。

表 15-5 介绍了准则到模式、反模式的映射。

表 15-5　准则到模式、反模式的映射

准则——如何度量	反模式——不该做的	模式——可以做的
开发人员长期不断地使用架构，减少了总成本和复杂性	简单复制并修改	由慢而快
架构小组明确理解关键最小需求并且将其构造成多应用共享的核心元素	缺乏有效抽象	迁移途径
通过长期的预算和行动确保当相关元素没有被共享、增加了不必要的复杂性时，或者是因为有明确的业务理由时，把相关元素从核心移走	编码大于架构	统计构件变更

准则 1：开发人员长期不断地使用架构，减少了总成本和复杂性。

使架构被正确地使用，需要获得并维持经理和实施人员的信任。当存在一个明晰的公共架构构想时，系统可以逐渐变得更加简单。Grady Boody 发现，“只有对一个系统的架构有清楚的理解，才能揭示公共的抽象和机制。”利用这种公共性能构造出更简单、更小和更可靠的系统。

与准则 1 相关的反模式和模式如下。

1）反模式：简单复制并修改

描述了当程序员在学会使用或重视架构之前被强迫迅速完成任务时发生的情况。他们不与构件负责人协商变更就复制并修改架构的部分代码，虽然复制提供了一个快速应付开发新特性的压力的方法，但是它通常会带来深远的后果。例如，如果在原始代码中发现了一个缺陷，组织怎样才能确保修正你和同事复制的所有代码呢?

在一个构件的生命周期中，有几个时机可以避免复制。鼓励工程师在复制构件之前先从构件负责人那里获得变更。可以把避免复制的推测方法加入编程风格指南，以便在代码复制期间识别和去除复制；如果有恰当的理由，这些推测方法可以允许有限的复制。也可以用自动化工具来识别复制代码，特别是在大型遗留系统中。当复制被识别后，可以用一些代码重组技术来去除重复的代码。

2）模式：由慢而快

描述了当开发人员为了跟上进度而拒绝使用架构结果却更慢时应该怎么做。解决方法是：放宽进度，加强过程。

让开发人员参与架构期望解决的问题的讨论，并通过开发部分解决方案来培训他们，给予过程比进度更高的优先级。指导开发人员逐步采用架构，把以前使用过这种过程有能力修改架构或过程来解决不同问题的专家介绍给开发团队，系统地、认真地遵循验证过程。

准则 2：架构小组明确理解关键最小需求，并且将其构造成多应用共享的核心元素。

与准则 2 相关的反模式与模式如下。

1）反模式：缺乏有效抽象

缺乏有效抽象是直接面对应用编程，虽然开始简洁，但随着应用发展，系统缺乏共享基础。该反模式描述了两种简化的努力走向极端的情况。榕树描述了长期建立单点解决方案的后果。单点解决方案通常是满足一个特定客户需求的最简单方法。根部肥大描述了一个架构或平台小组为平台所支持或可能支持的每个产品开发了专有的特性。

开发小组通过从头开发或者复制一个相关产品，然后根据当前问题进行修改来确保产品尽可能地简单。这样可以很快向客户提供初始产品。然而，这些产品没有共享任何东西。随着每个产品的维护和升级。榕树反模式开始出现，各个产品之间的分离越来越大。由于没有被一个共享平台强力支持，每一个分离产品都要求有自己的支撑结构，很像一棵榕树的分枝被很多枝蔓支撑。

对于这种情况，可以考虑采用类似先复制后合并模式的方法把很多为了适应特定功能特性而被修改的核心架构部分重新并入内核。如果先复制后合并和维护多个产品都不可行，就应考虑通过框架团队来建立一个共享平台。

根部肥大反模式则用枝少干粗的形象描述了这样一种情况，即一个架构或平台小组开发了太多针对单个客户的特性。结果共享了太多的功能，导致平台太大、太慢、推出太迟。根部肥大看起来就像一个倒立的马提尼酒杯，底部很大，杯口太小。

通过其他安排帮助产品小组开发不属于架构的产品特定模块，防止产品专有特性进入平台。把一个最小的共享特性集列入平台计划，并根据优先级以稳定的发布进度交付特性。

2）模式：迁移途径

迁移途径反应了在这样一种情形下的解决方案：架构师打算利用当前架构来支持一个新的有价值的应用领域。但是要在该新应用领域获得成功，需要当前用户所不具备的技能和观念，而拥有这些技能的用户团体却习惯于与当前平台不同的解决问题的方法。

对于这样一种情况，要选择一类最有可能扩大架构价值的采用者，并且努力使架构能被他们很快地理解和采用。考察所有类型的早期采用者，了解他们解决问题的方法和技能。确定哪一种类型最有可能理解或者预见到技术革新的成效，并且严格衡量该类型的用户是否具备解决方案所需的技能和知识。为有目标构想但缺乏重要技能集的专业人员提供迁移途径，提供一个简单的从平台获得基本成果的方法。然后，引导这些用户逐步更具体地使用平台。

准则 3：通过长期的预算和行动确保当相关元素没有被共享、增加了不必要的复杂性时，或者是因为有明确的业务理由时，把相关元素从核心移走。

改进一个架构需要时间和经费的稳定投入。稳定性确实很重要，因为当高级经理或主管最不愿意专注于架构时，也是架构最脆弱的时候。他们很容易被诱惑把架构师拉去参加一个紧急的项目以实现一个新特性，而使架构无人照看。

与准则 3 相关的反模式与模式如下。

1）反模式：编码大于架构

该反模式表明要防止架构师成为实现者。

首席架构师负责调整和维护架构，却被调动了工作要求竭尽全力地实现一个新特性集。这些特性实现了，架构小组却失去了领路人。因为没有时间对架构做出深思熟虑的改变，只好创建了架构的一个特殊版本来解决问题。结果新特性无法适合当前的架构。因为维护一个缺乏概念完整性的产品的工作量太大，结果问题越来越多。

为了防止出现这种情况，应该把首席架构师的时间合理分配给实现新特性和调整架构两个任务，让最能干的工程师来领导实现新特性。在提供时间和资源的同时，允许首席架构师指导实现，以使架构适应新的需求。

2）模式：统计构件变更

统计构件变更是一种通过观察不稳定程度来挑选需要调整的架构构件的方法。

如何才能知道应该重组（Refractor）什么——即从内核去除或简化什么呢？通过长期观测每个构件或子系统的不稳定程度，那些最不稳定的构件就是重组的候选者。因为不稳定表明构件是脆弱和不灵活的，因此应当根本改变该构件。也可以采用其他监控策略，例如监控讨论组以掌握经常被请求的构件。一名经验丰富的实施人员利用该方法可以很快确定哪些构件和子系统是简化的最佳目标，从而节约了时间和精力。

第 16 章　层次式架构设计

16.1　体系结构设计

1968 年，在 Garmish 召开的国际软件工程会议上，人们迫切地感到了软件危机给计算机软件产业的发展带来的巨大阻力。软件危机的两个比较大的问题是：软件的规模越来越大，软件复杂度越来越高。伴随着这两个问题的日益突出，整个软件系统结构的设计与规格说明便显得比算法选择和计算问题的数据结构更为重要。因此，代码级别的软件复用已经远远不能满足大型软件开发的需求，由此便引入了“软件体系结构”这一概念。

软件体系结构可定义为：软件体系结构为软件系统提供了结构、行为和属性的高级抽象，由构成系统的元素描述、这些元素的相互作用、指导元素集成的模式以及这些模式的约束组成。软件体系结构不仅指定了系统的组织结构和拓扑结构，并且显示了系统需求和构成系统的元素之间的对应关系，提供了一些设计决策的基本原理，是构建于软件系统之上的系统级复用。

软件体系结构贯穿于软件研发的整个生命周期内，具有重要的影响。这主要从以下三个方面来进行考察。

（1）利益相关人员之间的交流。软件体系结构是一种常见的系统抽象，代码级别的系统抽象仅仅可以成为程序员的交流工具，而包括程序员在内的绝大多数系统的利益相关人员都借助软件体系结构来作为相互沟通的基础。

（2）系统设计的前期决策。软件体系结构是我们所开发的软件系统最早期设计决策的体现，而这些早期决策对软件系统的后续开发、部署和维护具有相当重要的影响。这也是能够对系统进行分析的最早时间点。

（3）可传递的系统级抽象。软件体系结构是关于系统构造以及系统各个元素工作机制的相对较小、却又能够突出反映问题的模型。由于软件系统具有的一些共通特性，这种模型可以在多个系统之间传递，特别是可以应用到具有相似质量属性和功能需求的系统中，并能够促进大规模软件的系统级复用。

分层设计是一种最常见的架构设计方法，能有效地使设计简化，使设计的系统机构清晰，便于提高复用能力和产品维护能力。

16.2　表现层框架设计

16.2.1　使用 MVC 模式设计表现层

MVC 是一种目前广泛流行的软件设计模式。近年来，随着 J2EE（Java 2Enterprise Edition）的成熟，MVC 成为了 J2EE 平台上推荐的一种设计模式。MVC 强制性地把一个应用的输入、处理、输出流程按照视图、控制、模型的方式进行分离，形成了控制器、模型、视图三个核心模块。

（1）控制器（Controller）：接受用户的输入并调用模型和视图去完成用户的需求。该部分是用户界面与 Model 的接口。一方面它解释来自于视图的输入，将其解释成为系统能够理解的对象，同时它也识别用户动作，并将其解释为对模型特定方法的调用；另一方面，它处理来自于模型的事件和模型逻辑执行的结果，调用适当的视图为用户提供反馈。

（2）模型（Model）：应用程序的主体部分。模型表示业务数据和业务逻辑。一个模型能为多个视图提供数据。由于同一个模型可以被多个视图重用，所以提高了应用的可重用性。

（3）视图（View）：用户看到并与之交互的界面。视图向用户显示相关的数据，并能接收用户输入的数据，但是它并不进行任何实际的业务处理。视图可以向模型查询业务状态，但不能改变模型。视图还能接受模型发出的数据更新事件，从而对用户界面进行同步更新。

三者的协作关系如图 16-1 所示。

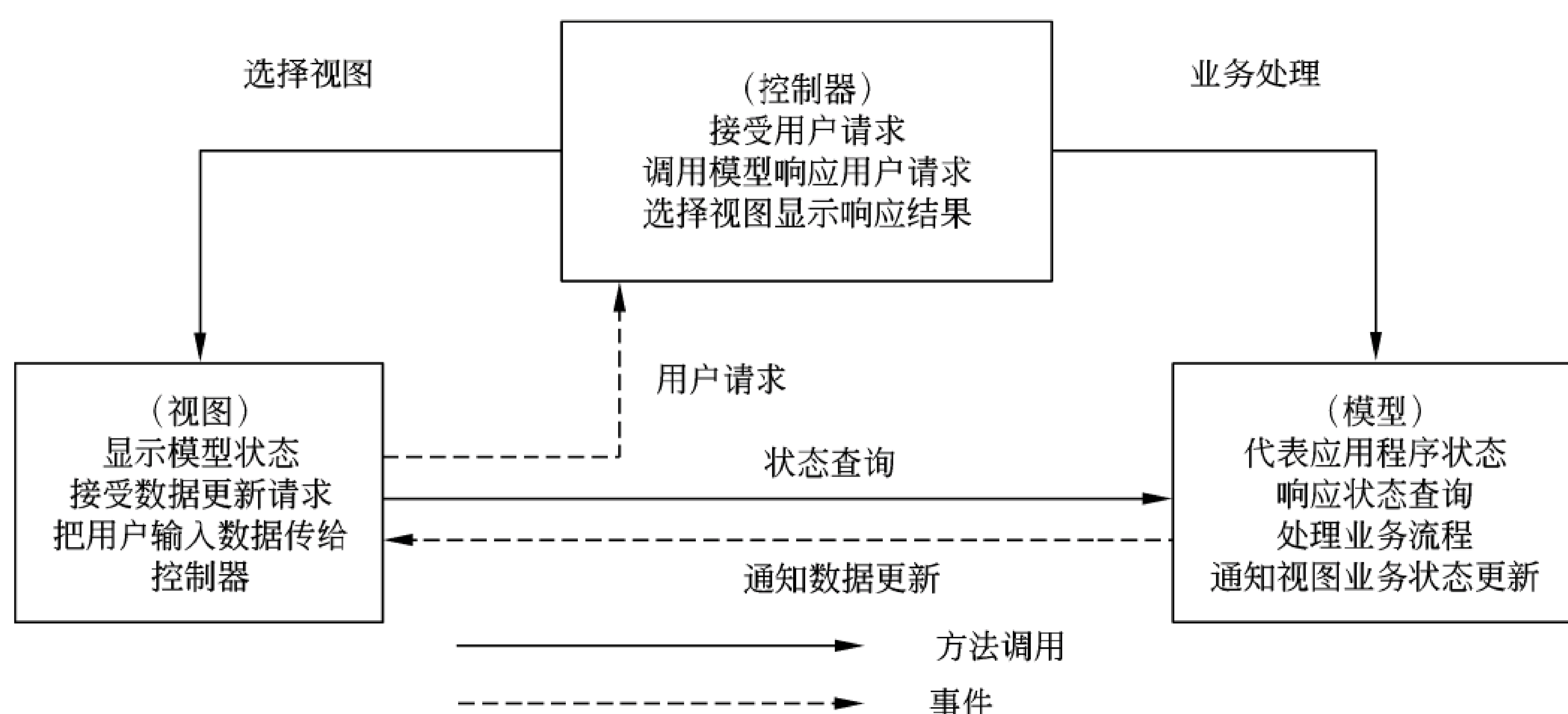


图 16-1　MVC 设计模式

从图 16-1 中可以看到，首先，控制器接收用户的请求，并决定应该调用哪个模型来处理；然后，模型根据用户请求进行相应的业务逻辑处理，并返回数据；最后，控制器调用相应的视图来格式化模型返回的数据，并通过视图呈现给用户。

使用 MVC 模式来设计表现层，可以有以下的优点。

（1）允许多种用户界面的扩展。在 MVC 模式中，视图与模型没有必然的联系，都是通过控制器发生关系，这样如果要增加新类型的用户界面，只需要改动相应的视图和控制器即可，而模型则无需发生改动。

（2）易于维护。控制器和视图可以随着模型的扩展而进行相应的扩展，只要保持一种公共的接口，控制器和视图的旧版本也可以继续使用。

（3）功能强大的用户界面。用户界面与模型方法调用组合起来，使程序的使用更清晰，可将友好的界面发布给用户。

MVC 是构建应用框架的一个较好的设计模式，可以将业务处理与显示分离，将应用分为控制器、模型和视图，增加了应用的可拓展性、强壮性及灵活性。基于 MVC 的优点，目前比较先进的 Web 应用框架都是基于 MVC 设计模式的。

16.2.2 使用 XML 设计表现层，统一 Web Form 与 Windows Form 的外观

XML（可扩展标记语言）与 HTML 类似，是一种标记语言。与主要用于控制数据的显示和外观的 HTML 标记不同，XML 标记用于定义数据本身的结构和数据类型。XML 已被公认为是优秀的数据描述语言，并且成为了业内广泛采用的数据描述标准。

由于 XML 的设计目标是描述数据并集中于数据的内容，所以虽然 XML 和 HTML 类似，但是业内很少采用 XML 作为表现层技术，表现层技术仍然是 HTML 唱主角。但是，由于 Web 应用程序对特定浏览器的局限以及性能问题，基于窗体表现形式的胖客户端应用程序又开始有了卷土重来的趋势。这两种应用程序各有优势，在未来很长一段时间这两种技术架构都会并存。因此，许多开发厂商在开发新产品时提出了既要支持胖客户端的表现形式，又要支持 Web 的表现形式。于是，有人提出将 GUI 用一个标准的形式描述，对于不同的表现形式，提供特定形式的转换器，根据 GUI 的描述转换成相应的表现形式。这就要求描述语言有非常好的通用性和扩展性，XML 恰恰是这种描述语言理想的载体。

对于大多数应用系统，GUI 主要是由 GUI 控件组成。控件可以看成是一个数据对象，其包含位置信息、类型和绑定的事件等。这些信息在 XML 中都可以作为数据结点保存下来，每一个控件都可以被描述成一个 XML 结点，而控件的那些相关属性都可以描述成这个 XML 结点的 Attribute。由于 XML 本身就是一种树型结构描述语言，所以可以很好地支持控件之间的层次结构。同时，XML 标记由架构或文档的作者定义，并且是无限制的，所以架构开发人员可以随意约定控件的属性，例如可以约定 type="button"是一个按钮，type="panel"是一个控件容器，type="Constraint"是位置等。这样，整个 GUI 就可

以完整而且简单地通过 XML 来描述。例如：

```
<component type="panel" constraint="16,22,78,200"/>
<component type="button" isvisible="false"
constraint="17,222,78,20"/>
</component>
```

这么一段 XML 很清晰地表示一个控件容器位置是（16,22,78,200），包含了一个不可视按钮。用上述的 XML 形式将 GUI 按照数据描述的形式保存下来代替原先特有的表现形式所需要的 GUI 描述载体。然后，对于特定的表现技术，实现不同的解析器解析 XML 配置文件。根据 XML 中的标签，按照特有的表现技术实例化的 GUI 控件实例对象。例如，解析器遇到 button，JFC 解析器会给予 JLabel 对象，XSLT 解析器会给予 <button id=… >这样一个 HTML 字符串，再调用特定表现技术的 API 将实例化出来的组件对象添加到 GUI 上显示。

从设计模式的角度来说，整个 XML 表现层解析的机制是一种策略模式。在调用显示 GUI 时，不是直接的调用特定的表现技术的 API，而是装载 GUI 对应的 XML 配置文件，然后根据特定的表现技术的解析器解析 XML，得到 GUI 视图实例对象。这样，对于 GUI 开发人员来说，GUI 视图只需要维护一套 XML 文件即可。

16.2.3　表现层中 UIP 设计思想

应用程序通常要用代码来管理用户界面，例如一个窗体可以决定下一个要呈现给用户的窗体。开发人员可以把这些代码写在 UI 代码中间，但是会使得代码复杂，不易复用、维护和扩展。另一方面，应用程序要运行在其他的平台也变得相当困难，因为它进行控制的逻辑和状态都不能被复用。

在大多数情况下，应用程序需要维护一个状态，如状态存储在窗体中，代码需要访问这个窗体以重新恢复状态。这样做会比较困难并且代码也会变得不雅，同时也会对用户接口的重用性和可扩展性产生影响。

用户应用系统的时候，他可能会先启动一个任务，离开一段时间后再回来继续。如果在中间用户关闭了应用程序，它将失去当前的状态，要想继续任务的话必须一切从头开始。因此设计程序的时候，必须分开来考虑工作流、导航、与商业服务的交互等各个组成部分，以获取数据并呈现给用户。

UIP（User Interface Process Application Block）是微软社区开发的众多 Application Block 中的其中之一，它是开源的。UIP 提供了一个扩展的框架，用于简化用户界面与商业逻辑代码的分离的方法，可以用它来写复杂的用户界面导航和工作流处理，并且它能够复用在不同的场景、并可以随着应用的增加而进行扩展。

使用 UIP 框架的应用程序把表现层分为了以下几层。

- User Interface Components：这个组件就是原来的表现层，用户看到的和进行交互都是这个组件，它负责获取用户的数据并且返回结果。
- User Interface Process Components：这个组件用于协调用户界面的各部分，使其配合后台的活动，例如导航和工作流控制，以及状态和视图的管理。用户看不到这一组件，但是这些组件为 User Interface Components 提供了重要的支持功能。

图 16-2 展示了这两层在基于.Net 的分布式应用程序中的位置。

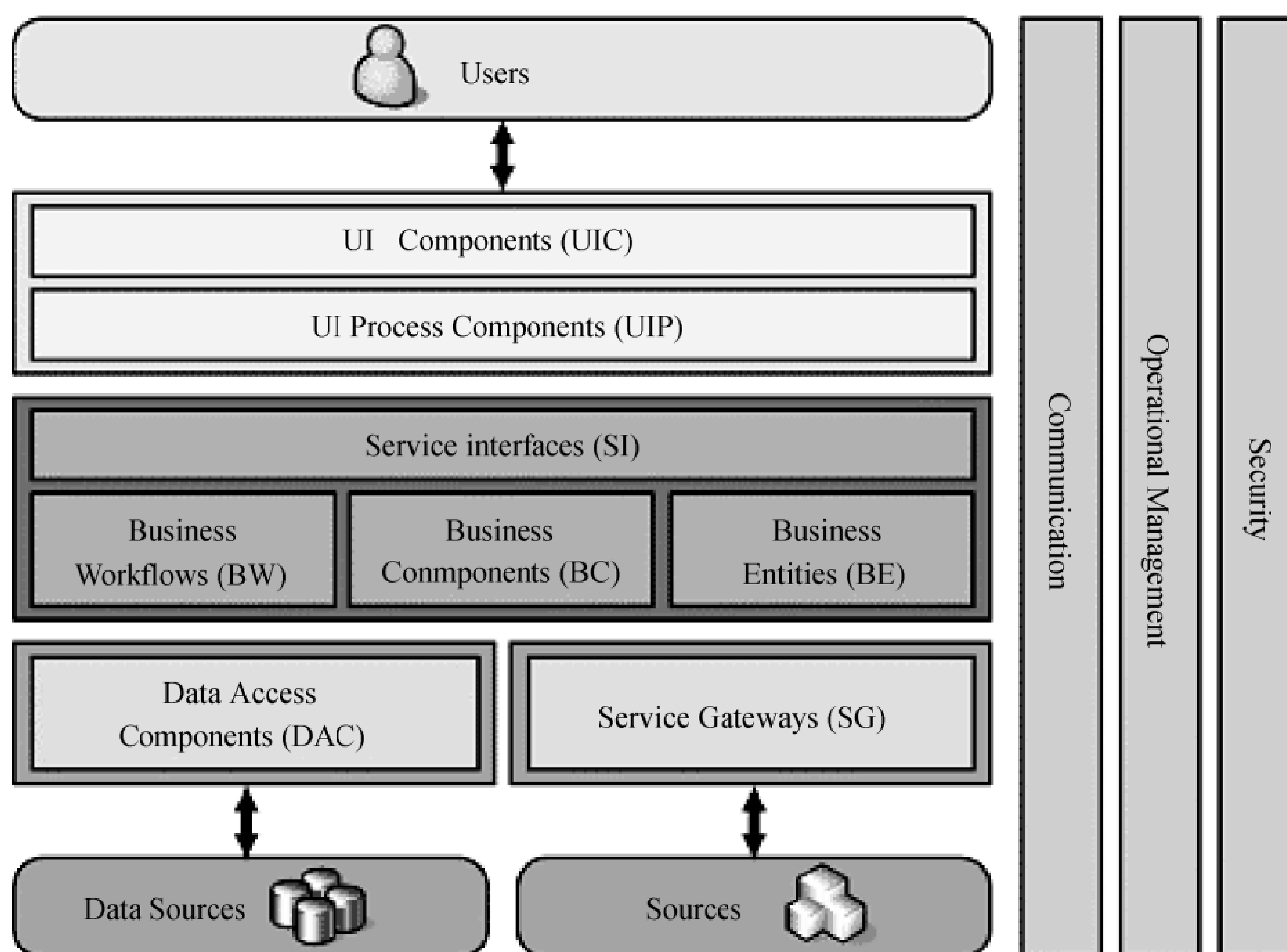


图 16-2 UI Components 和 UIP Components

UIP 的组件主要负责的功能是：管理经过 User Interface Components 的信息流；管理 UIP 中各个事件之间的事务；修改用户过程的流程以响应异常；将概念上的用户交互流程从实现或者涉及的设备上分离出来；保持内部的事务关联状态，通常是持有一个或者多个的与用户交互的事务实体。因此，这些组件也能进行从 UI 组件收集数据以执行服务器的成组的升级或是跟踪 UIP 中的任务过程的管理。

16.2.4 表现层动态生成设计思想

基于 XML 的界面管理技术可实现灵活的界面配置、界面动态生成和界面定制。其思路是用 XML 生成配置文件及界面所需的元数据，按不同需求生成界面元素及软件界面。

基于 XML 界面管理技术，包括界面配置、界面动态生成和界面定制三部分，如图 16-3 所示。

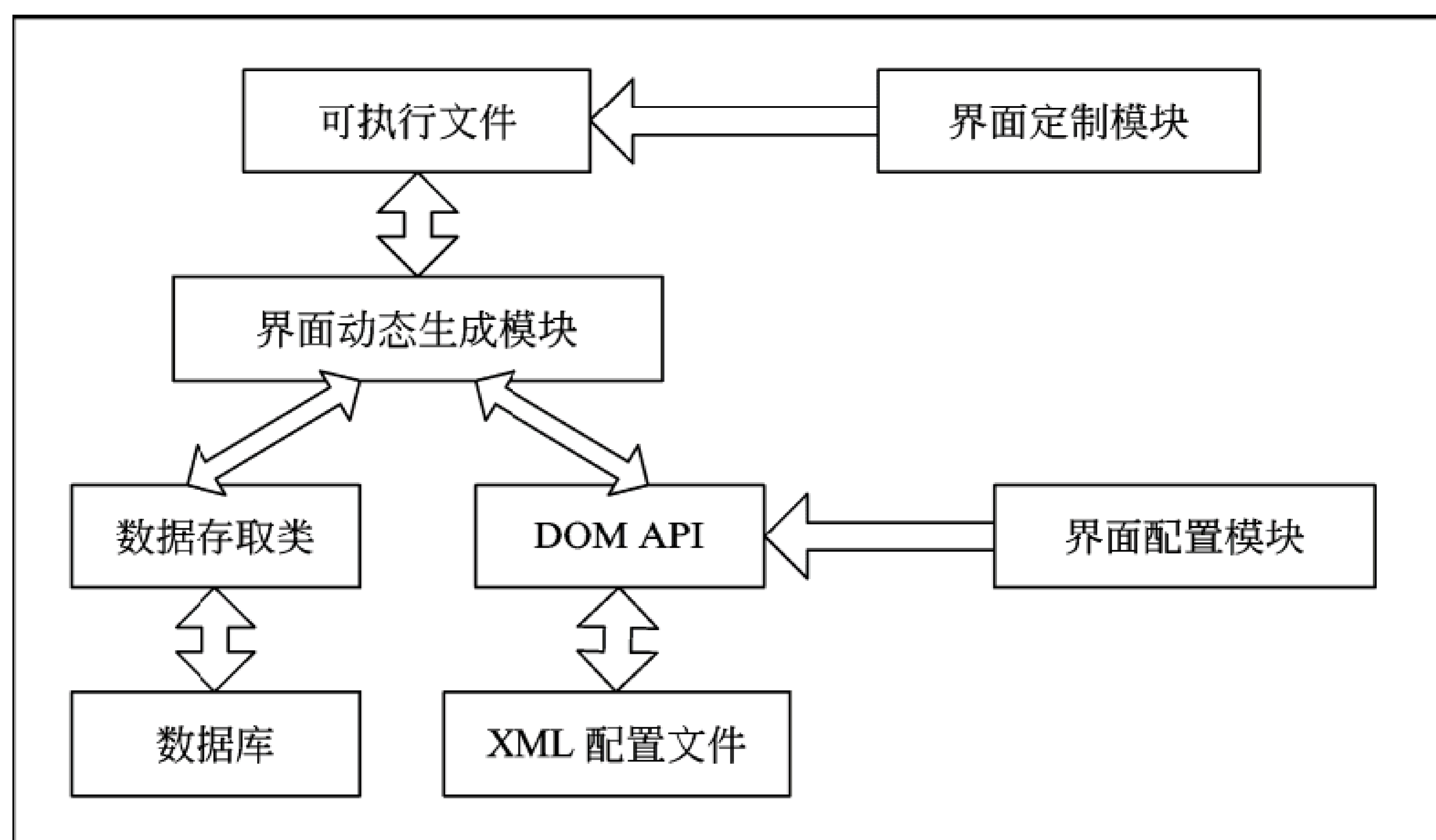


图 16-3　基于 XML 的界面管理技术框图

界面配置是对用户界面的静态定义，通过读取配置文件的初始值对界面配置。由界面配置对软件功能进行裁剪、重组和扩充，以实现特殊需求。

界面定制是对用户界面的动态修改过程，在软件运行过程中，用户可按需求和使用习惯，对界面元素（如菜单、工具栏、键盘命令）的属性（如文字、图标、大小和位置等）进行修改。软件运行结束，界面定制的结果被保存。

系统通过 DOM API 读取 XML 配置文件的表示层信息（初始界面大小、位置等），通过数据存取类读取数据库中的数据层信息，运行时由界面元素动态生成界面。界面配置和定制模块在软件运行前后修改配置文件、更改界面内容。

基于 XML 的界面管理技术实现的管理信息系统实现了用户界面描述信息与功能实现代码的分离，可针对不同用户需求进行界面配置和定制，能适应一定程度内的数据库结构改动。只需对 XML 文件稍加修改，即可实现系统的移植。

16.3　中间层架构设计

16.3.1　业务逻辑层组件设计

业务逻辑组件分为接口和实现类两个部分。

接口用于定义业务逻辑组件，定义业务逻辑组件必须实现的方法是整个系统运行的核心。通常按模块来设计业务逻辑组件，每个模块设计一个业务逻辑组件，并且每个业务逻辑组件以多个 DAO 组件作为基础，从而实现对外提供系统的业务逻辑服务。增加

业务逻辑组件的接口，是为了提供更好的解耦，控制器无须与具体的业务逻辑组件耦合，而是面向接口编程。

1．业务逻辑组件的实现类

业务逻辑组件以 DAO 组件为基础，必须接收 Spring 容器注入的 DAO 组件，因此必须为业务逻辑组件的实现类提供对应的 setter 方法。业务逻辑组件的实现类将 DAO 组件接口实例作为属性（面向接口编程），而对于复杂的业务逻辑，可能需要访问多个对象的数据，那么只需在这个方法里调用多个 DAO 接口，将具体实现委派给 DAO 完成。

2．业务逻辑组件的配置

由于业务逻辑组件的 DAO 组件从未被初始化过，那么业务方法如何完成？DAO 组件初始化是由 Spring 的反向控制（Inverse of Control，IoC）或者称为依赖注入（Dependency Injection，DI）机制完成的。为此，还需要在 applicationContext.xml 里面配置 FacadeManager 组件。

定义 FacadeManager 组件时必须为其配置所需要的 DAO 组件，配置信息表示 BaseManager 继承刚才配置的事务代理模板。并且由容器给 BaseManager 注入 dao 的组件，即 BaseDAOHibernate。而 target 则是 TransactionProxy FactoryBean 需要指定的属性，TransactionProxyFactoryBean 负责为某个 bean 实例生成代理，代理必须有个目标，target 属性则用于指定目标。

当然，也可以不使用事务代理模板及嵌套 bean，而是为组件指定单独的事务代理属性，让事务代理的目标引用容器中已经存在的 bean。

applicationContext.xml 文件的源代码配置了应用的数据源和 SessionFactory 等 bean，而业务逻辑组件也被部署在该文件中。

在配置文件中，采用继承业务逻辑组件的事务代理，将原有的业务逻辑组件作为嵌套 bean 配置，避免了直接调用没有事务特性的业务逻辑组件。

系统实现了所有的后台业务逻辑，并且向外提供了统一的 Facade 接口，前台 Web 层仅仅依赖这个 Facade 接口。这样，Web 层与后台业务层的耦合已经非常松散，系统可以在不同的 Web 框架中方便切换，即使将整个 Web 层替换掉也非常容易。

16.3.2 业务逻辑层工作流设计

工作流管理联盟（Workflow Management Coalition）将工作流定义为：业务流程的全部或部分自动化，在此过程中，文档、信息或任务按照一定的过程规则流转，实现组织成员间的协调工作以达到业务的整体目标。

工作流管理一直是企业界和学术界关注的热点领域。1993 年，国际上专门成立了工作流管理联盟（Workflow Management Coalition，WFMC），以便对工作流实现标准化管理。它是一种反映业务流程的计算机化的模型，是为了在先进计算机环境支持下实现经营过程集成与经营过程自动化而建立的可由工作流管理系统执行的业务模型。它解决的

主要问题是：使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者是促使此目标的实现。

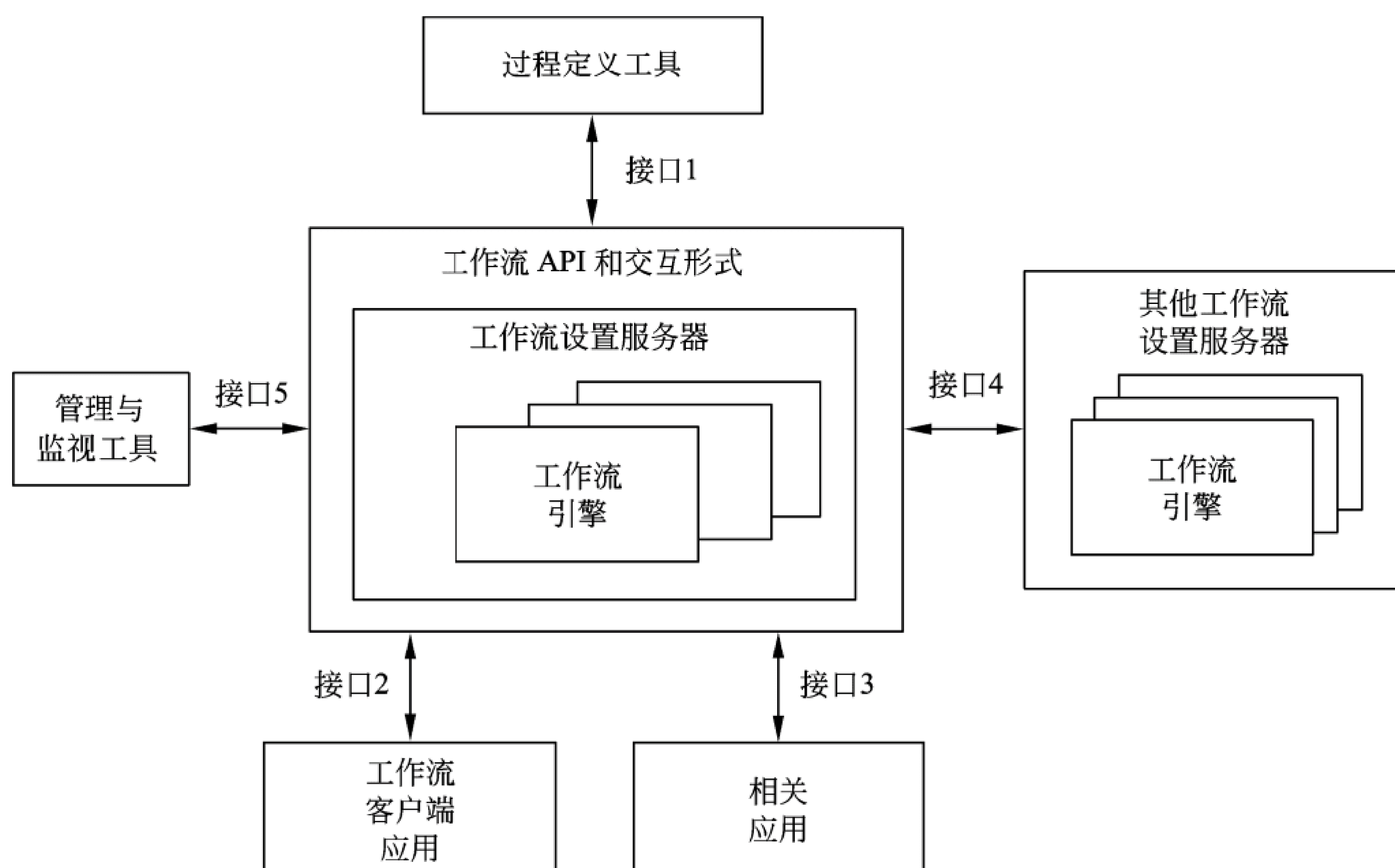


图 16-4　工作流参考模型

（1）interface 1：过程定义导入/导出接口。这个接口的特点是：转换格式和 API 调用，从而支持过程定义信息间的互相转换。这个接口也支持已完成的过程定义或过程定义的一部分之间的互相转换。早期标准是 WPDL，后来发展为 XPDL。

（2）interface 2：客户端应用程序接口。通过这个接口工作流机可以与任务表处理器交互，代表用户资源来组织任务。然后由任务表处理器负责，从任务表中选择、推进任务项。由任务表处理器或者终端用户来控制应用工具的活动。

（3）interface 3：应用程序调用接口。允许工作流机直接激活一个应用工具，来执行一个活动。典型的是调用以后台服务为主的应用程序，没有用户接口。当执行活动要用到的工具，需要与终端用户交互，通常是使用客户端应用程序接口来调用那个工具，这样可以为用户安排任务时间表提供更多的灵活性。

（4）interface 4：工作流机协作接口。其目标是定义相关标准，以使不同开发商的工作流系统产品相互间能够进行无缝的任务项传递。WFMC 定义了 4 个协同工作模型，包含多种协同工作能力级别。

（5）interface 5：管理和监视接口。提供的功能包括用户管理、角色管理、审查管理、资源控制、过程管理和过程状态处理器等。

用工作流的思想组织业务逻辑，优点是：将应用逻辑与过程逻辑分离，在不修改具

体功能的情况下，通过修改过程模型改变系统功能，完成对生产经营部分过程或全过程的集成管理，可有效地把人、信息和应用工具合理地组织在一起，发挥系统的最大效能。

16.3.3 业务逻辑层实体设计

业务逻辑层实体具有以下特点：业务逻辑层实体提供对业务数据及相关功能（在某些设计中）的状态编程访问。业务逻辑层实体可以使用具有复杂架构的数据来构建，这种数据通常来自数据库中的多个相关表。业务逻辑层实体数据可以作为业务过程的部分I/O 参数传递。业务逻辑层实体可以是可序列化的，以保持它们的当前状态。例如，应用程序可能需要在本地磁盘、桌面数据库（如果应用程序脱机工作）或消息队列消息中存储实体数据。业务逻辑层实体不直接访问数据库，全部数据库访问都是由相关联的数据访问逻辑组件提供的。 业务逻辑层实体不启动任何类型的事务处理，事务处理由使用业务逻辑层实体的应用程序或业务过程来启动。

在应用程序中表示业务逻辑层实体的方法有很多（从以数据为中心的模型到更加面向对象的表示法），如 XML、通用 DataSet、有类型的 DataSet 等。

以下示例显示了如何将一个简单的业务逻辑层实体表示为 XML。该业务逻辑层实体包含一个产品。

```
<?xml version="1.0"?>
<Product xmlns="urn:aUniqueNamespace">
<ProductID>1</ProductID>
<ProductName>Chai</ProductName>
<QuantityPerUnit>10 boxes x 20 bags</QuantityPerUnit>
<UnitPrice>18.00</UnitPrice>
<UnitsInStock>39</UnitsInStock>
<UnitsOnOrder>0</UnitsOnOrder>
<ReorderLevel>10</ReorderLevel>
</Product>
```

将业务逻辑层实体表示为 XML 的优点如下。

（1）标准支持。XML 是 World Wide Web Consortium（W3C）的标准数据表示格式。

（2）灵活性。XML 能够表示信息的层次结构和集合。

（3）互操作性。在所有平台上，XML 都是与外部各方及贸易伙伴交换信息的理想选择。

如果 XML 数据将由 ASP.NET 应用程序或 Windows 窗体应用程序使用，则还可以把这些 XML 数据装载到一个 DataSet 中，以利用 DataSet 提供的数据绑定支持。

将业务逻辑层实体表示为通用 DataSet。通用 DataSet 是 DataSet 类的实例，它是在 ADO.NET 的 System.Data 命名空间中定义的。DataSet 对象包含一个或多个 DataTable 对

象，用于表示数据访问逻辑组件从数据库检索到的信息。

图 16-5 所示为用于 Product 业务逻辑层实体的通用 DataSet 对象。该 DataSet 对象具有一个 DataTable，用于保存产品信息。该 DataTable 具有一个 UniqueConstraint 对象，用于将 ProductID 列标记为主键。DataTable 和 UniqueConstraint 对象是在数据访问逻辑组件中创建该 DataSet 时创建的。

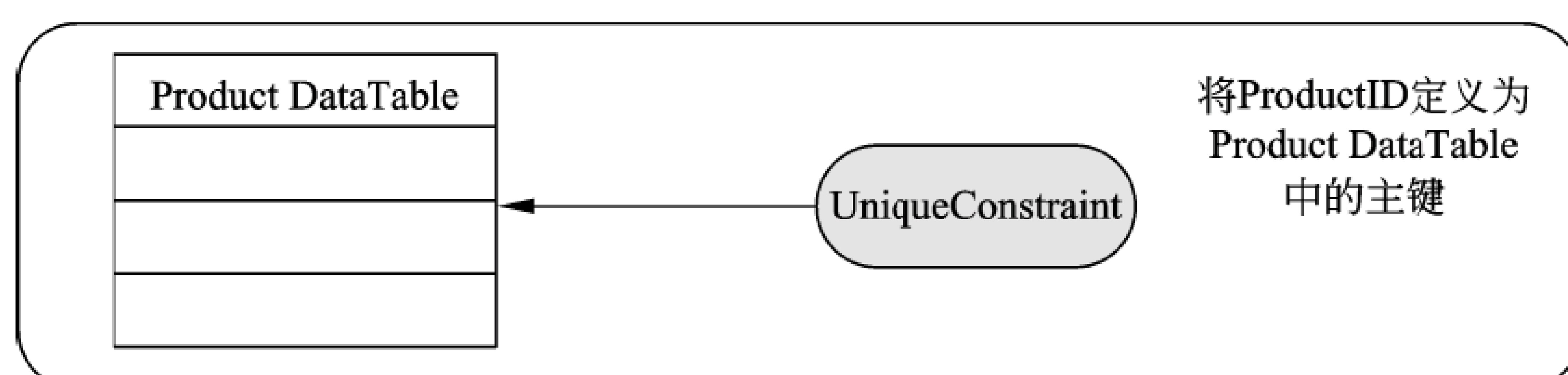


图 16-5　用于 Product 业务逻辑层实体的通用 DataSet

图 16-6 所示为用于 Order 业务逻辑层实体的通用 DataSet 对象。此 DataSet 对象具有两个 DataTable 对象，分别保存订单信息和订单详细信息。每个 DataTable 具有一个对应的 UniqueConstraint 对象，用于标识表中的主键。此外，该 DataSet 还有一个 Relation 对象，用于将订单详细信息与订单相关联。

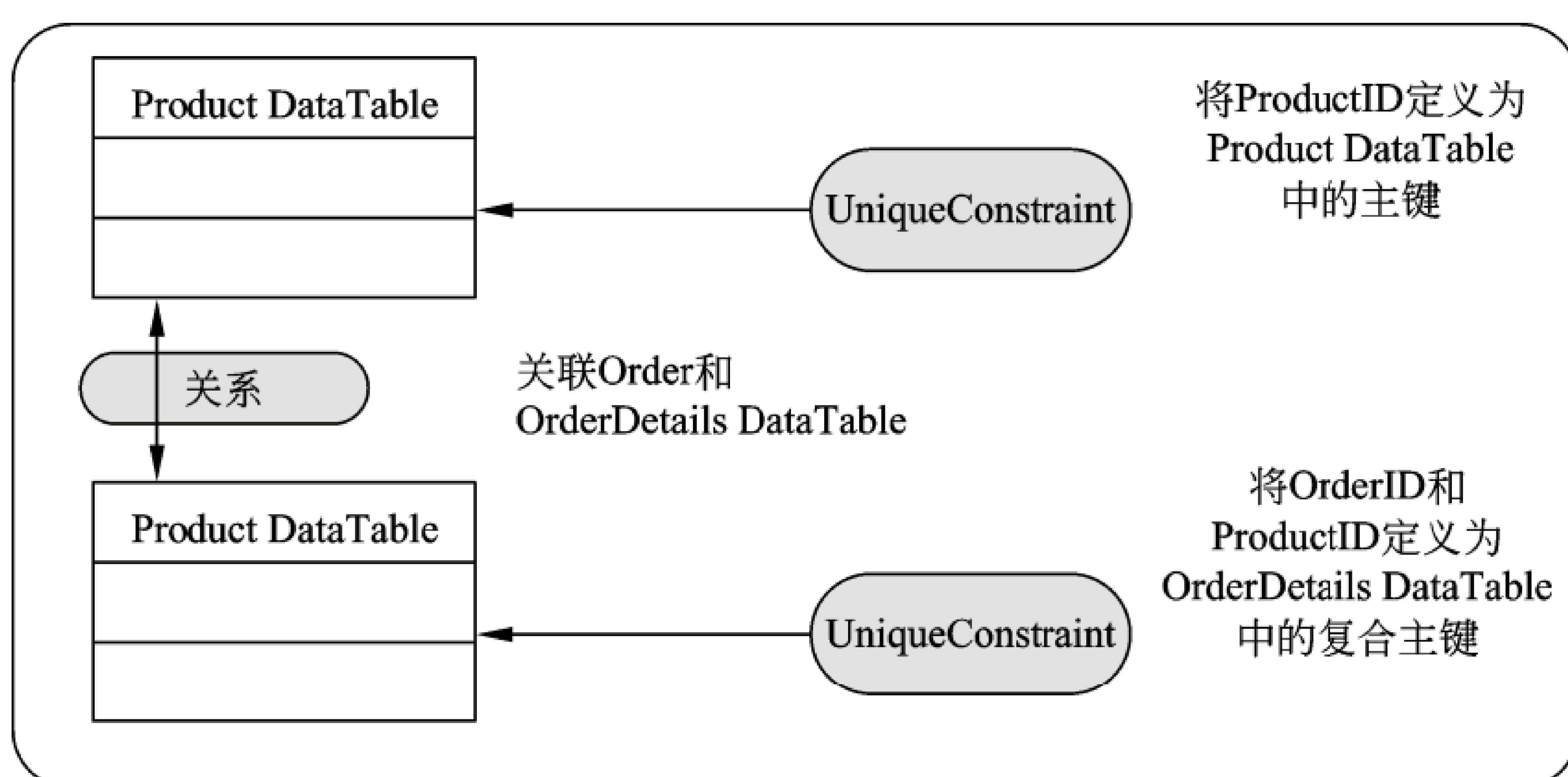


图 16-6　用于 Order 业务逻辑层实体的通用 DataSet

将业务逻辑层实体表示为通用 DataSet 的优点如下。

（1）灵活性。DataSet 可以包含数据的集合，能够表示复杂的数据关系。

（2）序列化。在层间传递时，DataSet 本身支持序列化。

（3）数据绑定。可以把 DataSet 绑定到 ASP.NET 应用程序和 Windows 窗体应用程序的任意用户界面控件。

（4）排序与过滤。可以使用 DataView 对象排序和过滤 DataSet。应用程序可以为同一个 DataSet 创建多个 DataView 对象，以便用不同方式查看数据。

（5）与 XML 的互换性。可以用 XML 格式读写 DataSet。

（6）开放式并发。在更新数据时，可以配合使用数据适配器与 DataSet 方便地执行开放式并发检查。

（7）可扩展性。如果修改了数据库架构，则适当情况下数据访问逻辑组件中的方法可以创建包含修改后的 DataTable 和 DataRelation 对象的 DataSet。

将业务逻辑层实体表示为有类型的 DataSet。有类型的 DataSet 是包含具有严格类型的方法、属性和类型定义以公开 DataSet 中的数据和元数据的类。

将业务逻辑层实体表示为有类型的 DataSet 的优点如下。

（1）代码易读。要访问有类型的 DataSet 中的表和列，可以使用有类型的方法和属性。

（2）有类型的方法和属性的提供使得使用有类型的 DataSet 比使用通用 DataSet 更方便。使用有类型的 DataSet 时，IntelliSense 将可用。

（3）编译时类型检查，无效的表名称和列名称将在编译时而不是在运行时检测。

16.3.4 业务逻辑层框架

业务框架位于系统架构的中间层，是实现系统功能的核心组件。采用容器的形式，便于系统功能的开发、代码重用和管理。图 16-7 便是在吸收了 SOA 思想之后的一个三层体系结构的简图。

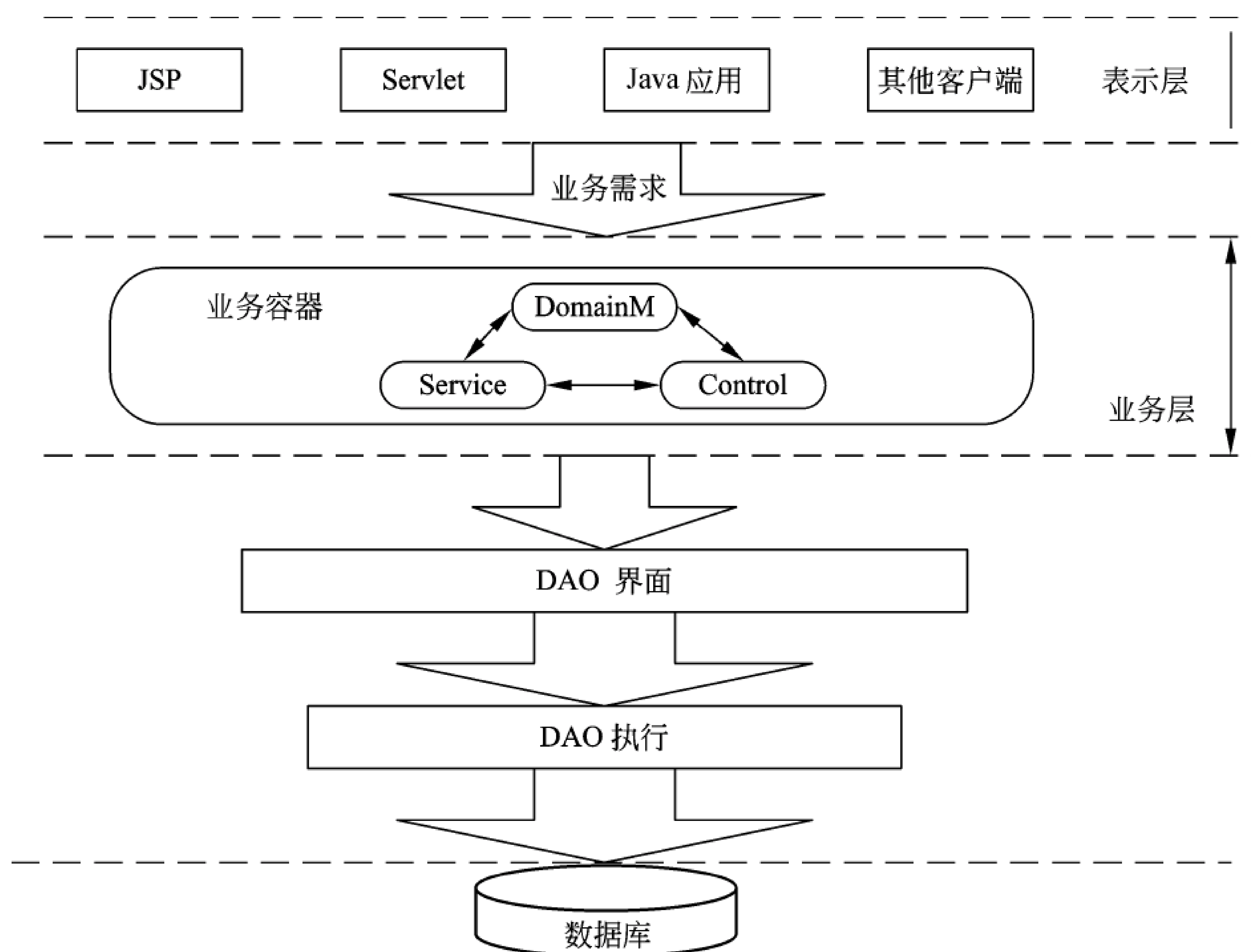


图 16-7 业务框架在整个系统架构中的位置

从图 16-7 中可以看到，业务层采用业务容器（Business Container）的方式存在于整个系统当中，采用此方式可以大大降低业务层和相邻各层的耦合，表示层代码只需要将业务参数传递给业务容器，便不需要业务层多余的干预。如此一来，可以有效地防止业务层代码渗透到表示层。

在业务容器中，业务逻辑是按照 Domain Model—Service—Control 思想来实现的。

（1）Domain Model 是领域层业务对象，它仅仅包含业务相关的属性。

（2）Service 是业务过程实现的组成部分，是应用程序的不同功能单元，通过在这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。这种具有中立的接口定义（没有强制绑定到特定的实现上）的特征称为服务之间的松耦合。松耦合系统的好处有两点，一是它的灵活性，二是当组成整个应用程序的每个服务的内部结构和实现逐渐地发生改变时，它能够继续存在。

（3）Control 服务控制器，是服务之间的纽带，不同服务之间的切换就是通过它来实现的。通过服务控制器控制服务切换可以将服务的实现和服务的转向控制分离，提高了服务实现的灵活性和重用性。

以下是 Domain Model-Service-Control 三者的互动关系。

（1）Service 的运行会依赖于 Domain Model 的状态，反之，Service 也会根据业务规则改变 Domain Model 的状态。

（2）Control 作为服务控制器，根据 Domain Model 的状态和相关参数决定 Service 之间的执行顺序及相互关系。

Domain Model—Service—Control 的互动关系，是吸取了 Model—View—Control 的优点，在“控制和显示的分离”的基础之上演变而来的，通过将服务和服务控制隔离，使程序具备高度的可重用性和灵活性。

16.4　数据访问层设计（持久层架构设计）

16.4.1　5 种数据访问模式

1．在线访问

在线访问是最基本的数据访问模式，也是在实际开发过程中最常采用的。

如图 16-8 所示，这种数据访问模式会占用一个数据库连接，读取数据，每个数据库操作都会通过这个连接不断地与后台的数据源进行交互。

2．Data Access Object

如图 16-9 所示，DAO 模式是标准 J2EE 设计模式之一，开发人员常常用这种模式将底层数据访问操作与高层业务逻辑分离开。

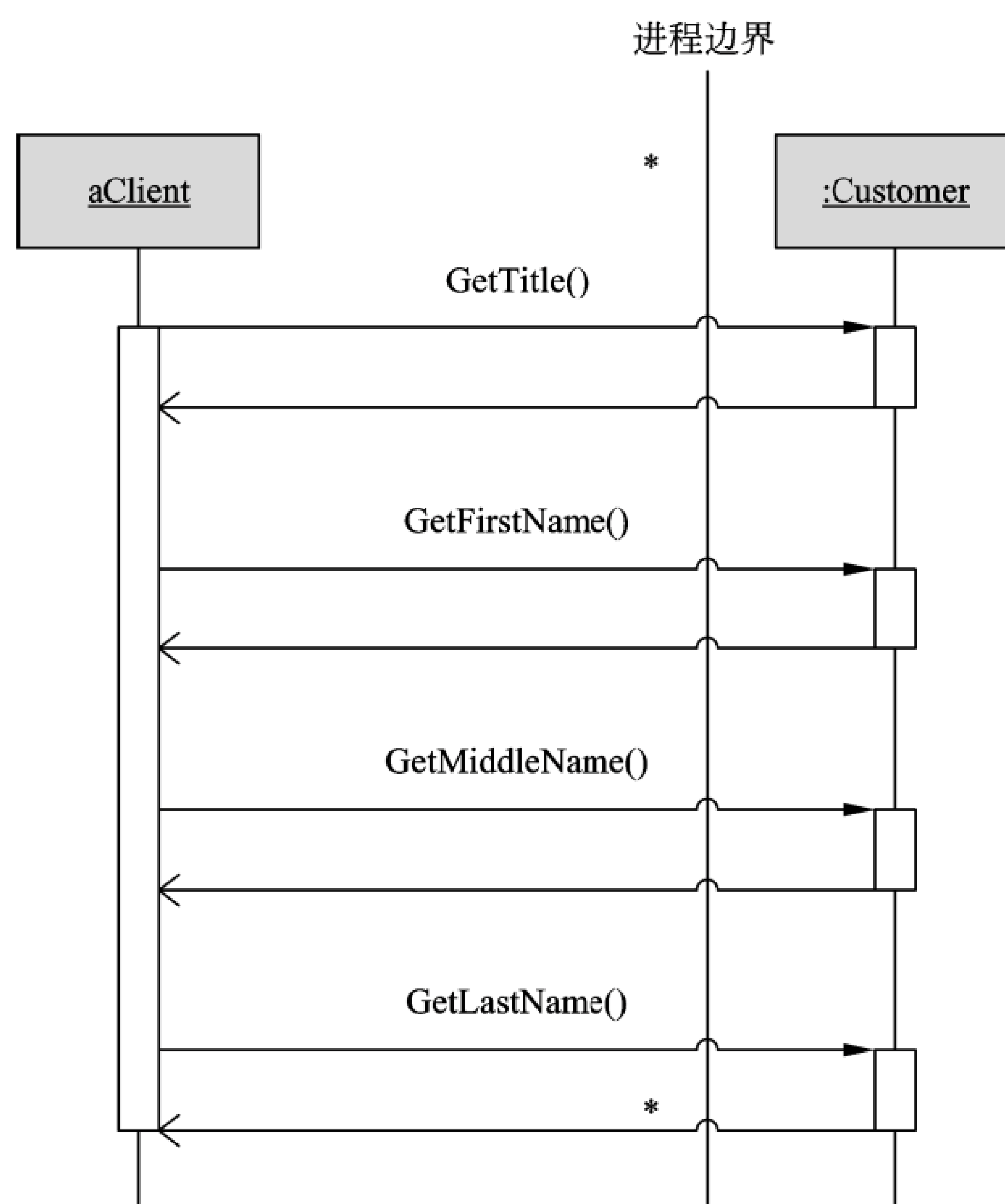


图 16-8　在线访问模式

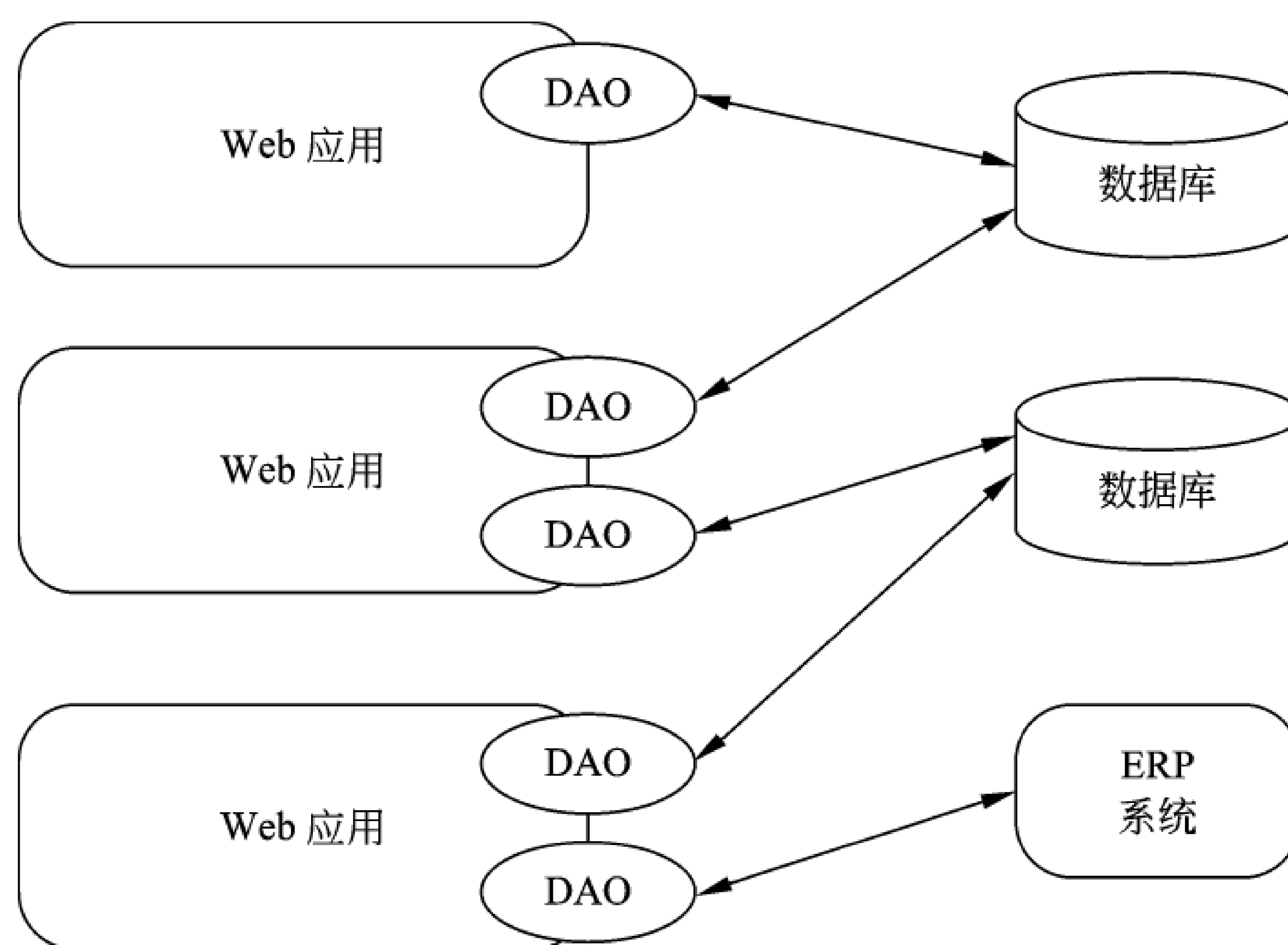


图 16-9　DAO 模式

一个典型的 DAO 实现通常有以下组件。

（1）一个 DAO 工厂类。

（2）一个 DAO 接口。

（3）一个实现了 DAO 接口的具体类。

（4）数据传输对象。

这当中具体的 DAO 类包含访问特定数据源的数据的逻辑。

3．Data Transfer Object

如图 16-10 所示，Data Transfer Object 是经典 EJB 设计模式之一。DTO 本身是这样一组对象或是数据的容器，它需要跨不同的进程或是网络的边界来传输数据。这类对象本身应该不包含具体的业务逻辑，并且通常这些对象内部只能进行一些诸如内部一致性检查和基本验证之类的方法，而且这些方法最好不要再调用其他的对象行为。

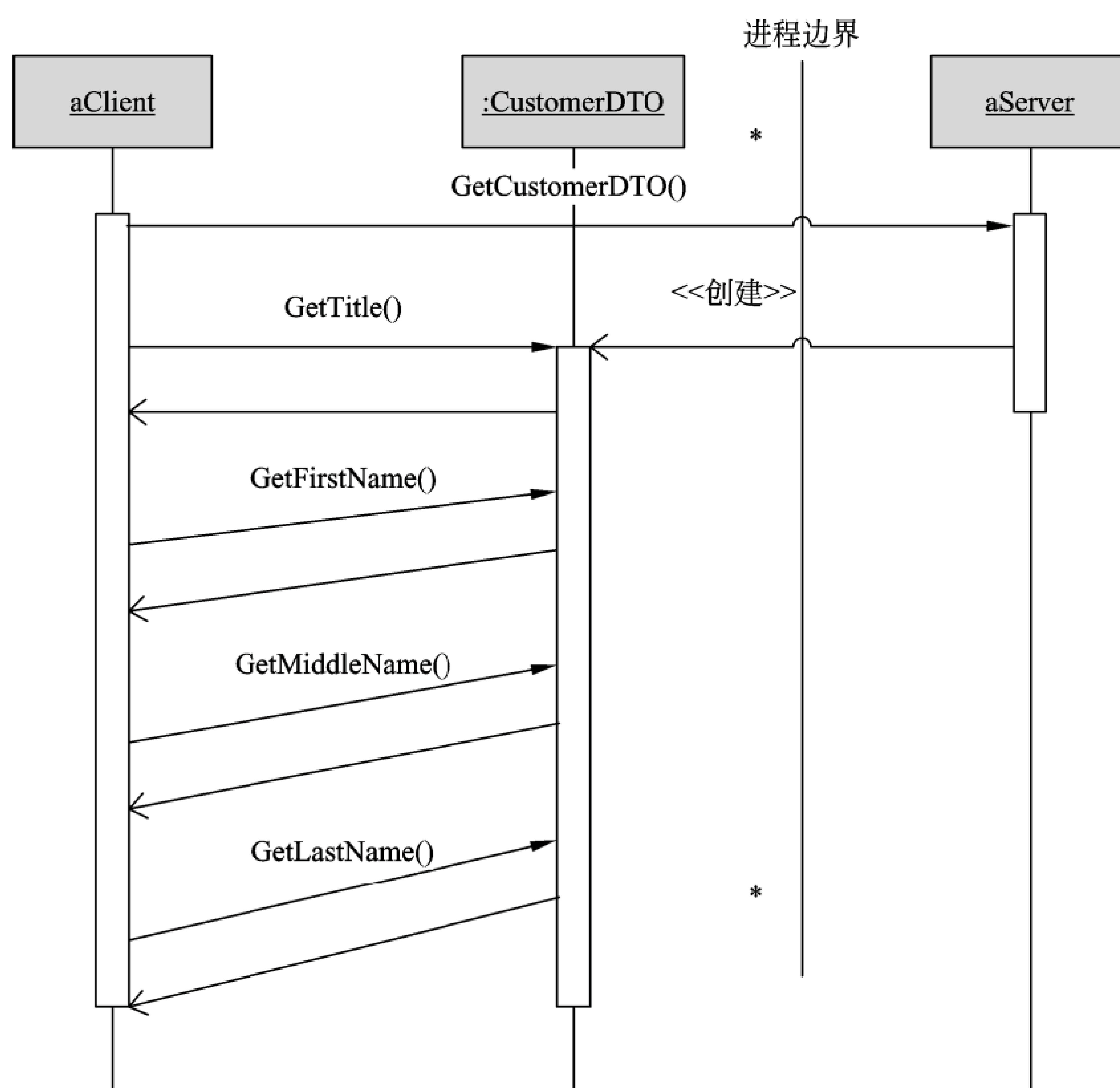


图 16-10　DTO 模式

在具体设计这类对象（DTO）时，通常可以有如下两种选择。

（1）使用编程语言内置的集合对象，它通常只需要一个类，就可以在整个应用程序中满足任何数据传输目的；而且几乎所有的编程语言都内置了集合类型，不需要再另外编写实现代码。同时，使用内置的集合对象来实现 DTO 对象的时候，客户端必须按位置序号（在简单数组的情况下）或元素名称（在键控集合的情况下）访问集合内的字段。

不过，集合存储的是同一类型（通常是最基本的 Object 类型）的对象，这有时会导致在编译时碰到一些无法检测到的编码错误。

（2）通过创建自定义类来实现 DTO 对象，通过定义显示的 get 或是 set 方法来访问数据。这种方式能够提供与任何其他对象完全一样的、客户端应用程序可访问的强类型对象。这种对象可以提供编译时的类型检查，但是增加了编码的工作量，若应用程序发出许多远程调用的话，需要编写大量的调用代码。

具体实现中有许多方法试图将上述这两种方法的优点结合在一起。第一种方法是代码生成技术，该技术可以生成脱离现有元数据（如可扩展标记语言 XML 架构）的自定义 DTO 类的源代码；第二种方法是提供更强大的集合，尽管它也是平台内置的一般的集合，但它将关系和数据类型信息与原始数据存储在一起，例如 IBM 提出的 SDO 技术或是微软 ADO.NET 中的 DataSet 就支持这类方法。

4. 离线数据模式

离线数据模式是以数据为中心，数据从数据源获取之后，将按照某种预定义的结构（这种结构可以是 SDO 中的 Data 图表结构，也同样可以是 ADO.NET 中的关系结构）存放在系统中，成为应用的中心。离线，对数据的各种操作独立于各种与后台数据源之间的连接或是事务；与 XML 集成，数据可以方便地与 XML 格式的文档之间互相转换；独立于数据源，离线数据模式的不同实现定义了数据的各异的存放结构和规则，这些都是独立于具体的某种数据源的。

5. 对象/关系映射（Object/Relation Mapping，O/R Mapping）

在最近几年，采用 OR 映射的指导思想来进行数据持久层的设计似乎已经成了一种潮流。对象/关系映射的基本思想来源于这样一种现实：大多数应用中的数据都是依据关系模型存储在关系型数据库中；而很多应用程序中的数据在开发或是运行时则是以对象的形式组织起来的。那么，对象/关系映射就提供了这样一种工具或是平台，能够帮助将应用程序中的数据转换成关系型数据库中的记录；或是将关系数据库中的记录转换成应用程序中代码便于操作的对象。

16.4.2 工厂模式在数据访问层应用

在应用程序的设计中，数据库的访问是非常重要的，数据库的访问需要良好的封装性和可维护性。在.Net 中，数据库的访问，对于微软自家的 SqlServer 和其他数据库（支持 OleDb），采用不同的访问方法，这些类分别分布于 System.Data.SqlClient 和 System.Data.OleDb 名称空间中。微软后来又推出了专门用于访问 Oracle 数据库的类库。我们希望在编写应用系统的时候，不因这么多类的不同而受到影响，尽量做到数据库无关。

这就需要在实际开发过程中将这些数据库访问类再作一次封装。经过这样的封装，不仅可以达到上述的目标，还可以减少操作数据库的步骤，减少代码编写量。工厂设计

模式是使用的主要方法。

工厂模式定义一个用于创建对象的接口，让子类决定实例化哪一个类。工厂方法使一个类的实例化延迟到其子类。这里可能会处理对多种数据库的操作，因此，需要首先定义一个操纵数据库的接口，然后根据数据库的不同，由类工厂决定实例化哪个类。

下面首先来定义这个访问接口。为了方便说明问题，在这里只列出了比较少的方法，其他的方法是很容易参照添加的。

```
public interface DataAccess
{
    DatabaseType DatabaseType{get;}                //数据库类型
   IDbConnection DbConnection{get;}                //得到数据库连接
    void Open();                                   //打开数据库连接
    void Close();                                  //关闭数据库连接
    IDbTransaction BeginTransaction();             //开始一个事务
    int ExecuteNonQuery(string commandText);       //执行 Sql 语句
    DataSet ExecuteDataset(string commandText);    //执行 Sql,返回 DataSet
}
```

因为 DataAccess 的具体实现类有一些共同的方法，所以先从 DataAccess 实现一个抽象的 AbstractDataAccess 类，包含一些公用方法。然后，分别为 Sql Server、Oracle 和 OleDb 数据库编写三个数据访问的具体实现类。

```
public sealed class MSSqlDataAccess : AbstractDataAccess
    {
        …//具体实现代码
    }

    public class OleDbDataAccess : AbstractDataAccess
    {
    …//具体实现代码
    }

    public class OracleDataAccess : AbstractDataAccess
    {
        …//具体实现代码
    }
```

现在已经完成了所要的功能，下面需要创建一个 Factory 类，来实现自动数据库切换的管理。这个类很简单，主要的功能就是根据数据库类型，返回适当的数据库操纵类。

```
public sealed class DataAccessFactory
```

```
{
    private DataAccessFactory(){}
    private static PersistenceProperty defaultPersistenceProperty;
    public static PersistenceProperty DefaultPersistenceProperty
    {
        get{return defaultPersistenceProperty;}
        set{defaultPersistenceProperty=value;}
    }
    public static DataAccess CreateDataAccess(PersistenceProperty pp)
    {
        DataAccess dataAccess;
        switch(pp.DatabaseType)
        {
            case(DatabaseType.MSSQLServer):
                dataAccess = new MSSqlDataAccess(pp.Connection-
                String);
                break;
            case(DatabaseType.Oracle):
                dataAccess = new OracleDataAccess(pp.Connection-
                String);
                break;
            case(DatabaseType.OleDBSupported):
                dataAccess = new OleDbDataAccess(pp.ConnectionString);
                break;
            default:
                dataAccess=new MSSqlDataAccess(pp.ConnectionString);
                break;
        }
        return dataAccess;
    }
    public static DataAccess CreateDataAccess()
    {
        return CreateDataAccess(defaultPersistenceProperty);
    }
}
```

现在一切都完成了，客户端在代码调用的时候，可能就是采用如下形式。

```
PersistenceProperty pp = new PersistenceProperty();
pp.ConnectionString = "server=127.0.0.1;uid=sa;pwd=;database=Northwind;";
pp.DatabaseType = DatabaseType. MSSQLServer;
```

```
pp.UserID = "sa";
pp.Password = "";
DataAccess db= DataAccessFactory.CreateDataAccess(pp)
db.Open();
…//db.需要的操作
db.Close();
```

或者，如果事先设定了 DataAccessFactory 的 DefaultPersistenceProperty 属性，可以直接使用 DataAccess db= DataAccessFactory.CreateDataAccess()方法创建 DataAccess 实例。

当数据库发生变化时，只需要修改 PersistenceProperty 的值，客户端不会感觉到变化，也不用去关心。这样，实现了良好的封装性。当然，前提是你在编写程序时，没有用到特定数据库的特性，例如，Sql Server 的专用函数。

16.4.3　ORM、Hibernate 与 CMP2.0 设计思想

ORM（Object-Relation Mapping）在关系型数据库和对象之间作一个映射，这样，在具体操作数据库时，就不需要再去和复杂的 SQL 语句打交道，只要像平时操作对象一样操作即可。

当你开发一个应用程序的时候（不使用 OR Mapping），可能会涉及许多数据访问层的代码，用来从数据库保存、删除和读取对象信息等，然而这些代码写起来总是重复的。

一个更好的办法就是引入 OR Mapping。实质上，一个 OR Mapping 会为你生成 DAL。与其自己写 DAL 代码，不如用 OR Mapping，你只需要关心对象就好。

使用 ORM 可以大大降低学习和开发成本。而在实际的开发中，真正对客户有价值的是其独特的业务功能，而不应该把大量时间花费在编写数据访问、CRUD 方法、后期的 Bug 查找和维护上。在使用 ORM 之后，ORM 框架已经把数据库转变成了我们熟悉的对象，我们只需要了解面向对象开发就可以实现数据库应用程序的开发，不需要浪费时间在 SQL 上。同时也可减少代码量，减少数据层出错机会。

通过 Cache 的实现，能够对性能进行调优，实现了 ORM 区隔了实际数据存储和业务层之间的关系，能够对每一层进行单独跟踪，增加了性能优化的可能。

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了轻量级的对象封装，使 Java 程序员可以随心所欲地使用对象编程思维来操纵数据库。它不仅提供了从 Java 类到数据表之间的映射，还提供了数据查询和恢复机制。相对于使用 JDBC 和 SQL 来手工操作数据库，Hibernate 可以大大减少操作数据库的工作量。另外，Hibernate 可以利用代理模式来简化载入类的过程，这将大大减少利用 Hibernate QL 从数据库提取数据的代码的编写量。Hibernate 可以和多种 Web 服务器或者应用服务器良好集成，如今已经支持几乎所有流行的数据库服务器。

Hibernate 技术本质上是一个提供数据库服务的中间件，它的架构如图 16-11 所示。

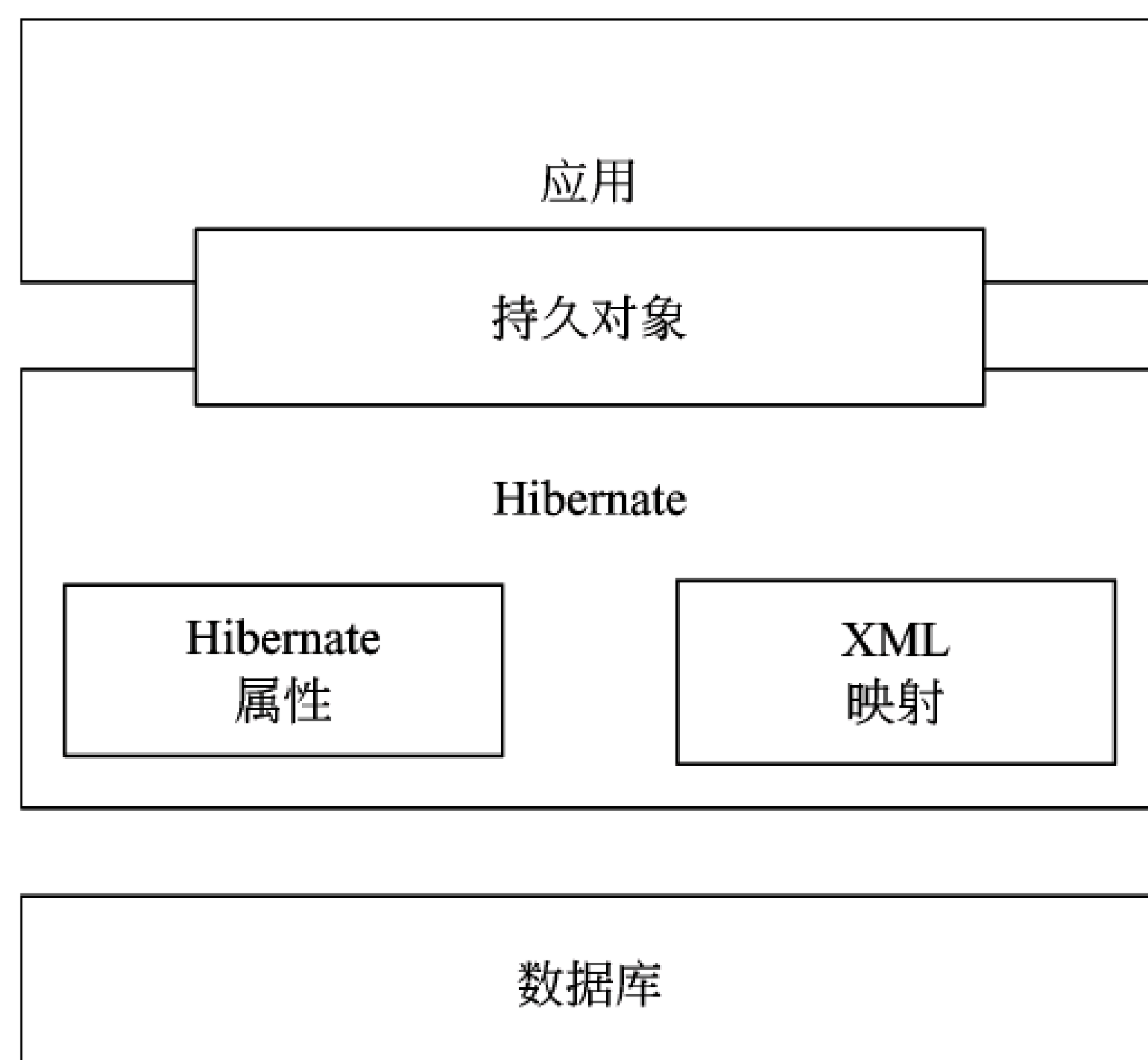


图 16-11　Hibernate 架构图

图 16-11 显示了 Hibernate 件（如 hibernate.properties）的工作原理，它是利用数据库以及其他一些配置 XML Mapping 等来为应用程序提供数据持久化服务的。

Hibernate 具有很大的灵活性，但同时它的体系结构比较复杂，提供了好几种不同的运行方式。在轻型体系中，应用程序提供 JDBC 连接，并且自行管理事务，这种方式使用了 Hibernate 的一个最小子集。在全面解决体系中，对于应用程序来说，所有底层的 JDBC/JTA API 都被抽象了，Hibernate 会替你照管所有的细节。

Hibernate 是一个功能强大，可以有效地进行数据库数据到业务对象的 0/R 映射方案。Hibernate 推动了基于普通 Java 对象模型，用于映射底层数据结构的持久对象的开发。通过将持久层的生成自动扩展到一个更大的范围，Hibernate 使开发人员专心实现业务逻辑而不用分心于繁琐的数据库方面的逻辑，同时提供了更加合理的模块划分的方法。

16.4.4　灵活运用 Xml Schema

XML Schema 用来描述 XML 文档合法结构、内容和限制。XML Schema 由 XML 1. 0 自描述，并且使用了命名空间，有丰富的内嵌数据类型及其强大的数据结构定义功能，充分地改造了并且极大地扩展了 DTDs（传统描述 XML 文档结构和内容限制的机制）的能力，将逐步替代 DTDs，成为 XML 体系中正式的类型语言，同 XML 规范、Namespace 规范一起成为 XML 体系的坚实基础。

XML Schema 由诸如类型定义和元素声明的组件组成，可以用来评估一个格式良好元素和属性信息的有效性。XML Schema 是 Schema 组件的集合，这些组件分为三组：基本组件、组件和帮助组件。其中基本组件包括简单类型定义、复杂类型定义、属性声

明和元素声明；组件包括属性组、完整性约束定义、模型组和符号声明；帮助组件包括注释、模型组、小品词、通配符和属性使用。Schema 组件详细说明了抽象数据模型的每个组件的严格语义，每个组件在 XML 中的表示，一个 XML Schema 文档类型的 DTD 和 XML Schema 引用。

XML Schema 提供了创建 XML 文档必要的框架，详细说明了一个 XML 文档的不同元素和属性的有效结构、限制和数据类型。XML Schema 规范由如下三部分组成。

（1）XML Schema PartO：Primer。一个非标准化的文档，提供了 XML Schema 的一个简单可读的描述，目的是快速地理解如何利用 XML Schema 语言创建一个 Schema（框架）。

（2）XML Schema Partl：Structures。这一部分详细说明了 XML Schema 定义语言，这个语言为描述 XML 1.0 文档的结构和内容限制提供了便利，包括开发了 XML Namespace（命名空间）的使用。

（3）XML Schema Part2：Datatypes。这一部分定义了可用于 XML Schema 和其他 XML 规范中的定义数据类型的方法。这个数据类型语言，本身由 XML 1. 0 自描述，提供了说明元素和属性数据类型的 XML 1. 0 文档类型定义（DTDs）的一个超集。这部分提出了标准的数据类型内容集合，其中讲述了目的、需求、范围和术语。XML Schema 与 DTD 相比，有其独特的特点，提供了丰富的数据类型，实现了继承和复用，与命名空间紧密联系，易于使用。

与 DTD 不同，XML Schema 规范提供了丰富的数据类型。其中不仅包括一些内嵌的数据类型，如 string、integer、Boolean、time 和 date 等，还提供了定义新类型的能力，如 complexType 和 simpleType。开发者可以利用内嵌的数据类型和用户定义的数据类型，有效地定义和限制 XML 文档的属性和元素值。

XML Schema 支持继承是它的另一特点。可以利用从已经存在的 schema 中获得某些类型而构造新的 schema，也可以在不需要时使获得的类型无效。同时，XML Schema 能将一个 schema 分成单独的组件，这样，在写 Schema 时，就可以正确地引用已经定义的组件。继承性使得软件复用更加有效，帮助开发者避免了每一次创建都要从零开始，极大地提高了软件开发和维护的效率。

XML Schema 与 XML Namespace 紧密联系，使得在一个命名空间中创建元素和属性非常容易。这种联系简化了使用多个命名空间定义多个 schema 的 XML 文档的创建和验证文档有效性。

16.4.5　事务处理设计

事务是现代数据库理论中的核心概念之一。如果一组处理步骤或者全部发生或者一步也不执行，我们称该组处理步骤为一个事务。当所有的步骤像一个操作一样被完整地执行，我们称该事务被提交。由于其中的一部分或多步执行失败，导致没有步骤被提交，

则事务必须回滚（回到最初的系统状态）。事务必须服从 ISO/IEC 所制定的 ACID 原则。ACID 是原子性（atomicity）、一致性（consistency）、隔离性（isolation）和持久性（durability）的缩写。事务的原子性表示事务执行过程中的任何失败都将导致事务所做的任何修改失效。一致性表示当事务执行失败时，所有被该事务影响的数据都应该恢复到事务执行前的状态。隔离性表示在事务执行过程中对数据的修改，在事务提交之前对其他事务不可见。持久性表示已提交的数据在事务执行失败时，数据的状态都应该正确。

一般情况下，J2EE 应用服务器支持 JDBC 事务、JTA（Java Transaction API）事务和容器管理事务。一般情况下，最好不要在程序中同时使用上述三种事务类型，例如在 JTA 事务中嵌套 JDBC 事务。另外，事务要在尽可能短的时间内完成，不要在不同方法中实现事务的使用。下面举列说明两种事务处理方式。

1．JavaBean 中使用 JDBC 方式进行事务处理

在 JDBC 中怎样将多个 SQL 语句组合成一个事务呢？在 JDBC 中，打开一个连接对象 Connection 时，默认是 auto-commit 模式，每个 SQL 语句都被当作一个事务，即每次执行一个语句，都会自动地得到事务确认。为了能将多个 SQL 语句组合成一个事务，要将 auto-commit 模式屏蔽掉。在 auto-commit 模式屏蔽掉之后，如果不调用 commit()方法，SQL 语句不会得到事务确认。在最近一次 commit()方法调用之后的所有 SQL 会在方法 commit()调用时得到确认。

```
public int delete(int sID) {
 dbc = new DataBaseConnection();
 Connection con = dbc.getConnection();
 try {
  con.setAutoCommit(false); //更改 JDBC 事务的默认提交方式
  dbc.executeUpdate("delete from bylaw where ID=" + sID);
  dbc.executeUpdate("delete from bylaw _content where ID=" + sID);
  dbc.executeUpdate("delete from bylaw _affix where bylawid=" + sID);
  con.commit();             //提交 JDBC 事务
  con.setAutoCommit(true);  //恢复 JDBC 事务的默认提交方式
  dbc.close();
  return 1;
 }
 catch (Exception exc) {
  con.rollBack();           //回滚 JDBC 事务
  exc.printStackTrace();
  dbc.close();
  return -1;
  }
}
```

2. SessionBean 中的 JTA 事务

JTA 是事务服务的 J2EE 解决方案。本质上，它是描述事务接口（例如 UserTransaction 接口，开发人员直接使用该接口或者通过 J2EE 容器使用该接口来确保业务逻辑能够可靠地运行）的 J2EE 模型的一部分。JTA 具有的三个主要的接口，分别是 UserTransaction 接口、TransactionManager 接口和 Transaction 接口。这些接口共享公共的事务操作，例如 commit()和 rollback()；但是也包含特殊的事务操作，例如 suspend()、resume()和 enlist()，它们只出现在特定的接口上，以便在实现中允许一定程度的访问控制。例如，UserTransaction 能够执行事务划分和基本的事务操作，而 TransactionManager 能够执行上下文管理。

应用程序可以调用 UserTransaction.begin()方法开始一个事务，该事务与应用程序正在其中运行的当前线程相关联。底层的事务管理器实际处理线程与事务之间的关联。UserTransaction.commit()方法终止与当前线程关联的事务。UserTransaction.rollback()方法将放弃与当前线程关联的当前事务。

```
 public int delete(int sID) {
DataBaseConnection dbc = null;
dbc = new DataBaseConnection();
dbc.getConnection();
UserTransaction transaction = sessionContext.getUserTransaction();
                                          //获得 JTA 事务
try {
 transaction.begin();                     //开始 JTA 事务
 dbc.executeUpdate("delete from bylaw where ID=" + sID);
 dbc.executeUpdate("delete from bylaw _content where ID=" + sID);
 dbc.executeUpdate("delete from bylaw _affix where bylawid=" + sID);
 transaction.commit();                    //提交 JTA 事务
 dbc.close();
 return 1;
}
catch (Exception exc) {
 try {
  transaction.rollback();                 //JTA 事务回滚
 }
 catch (Exception ex) {
  //JTA 事务回滚出错处理
  ex.printStackTrace();
 }
 exc.printStackTrace();
 dbc.close();
```

```
        return -1;
    }
}
```

16.4.6 连接对象管理设计

在基于 JDBC 的数据库应用开发中，数据库连接的管理是一个难点，因为它是决定该应用性能的一个重要因素。

对于共享资源，有一个很著名的设计模式——资源池。该模式正是为了解决资源频繁分配、释放所造成的问题。把该模式应用到数据库连接管理领域，就是建立一个数据库连接池，提供一套高效的连接分配、使用策略。

建立连接池的第一步，就是要建立一个静态的连接池。所谓静态，是指池中的连接是在系统初始化时就分配好的，并且不能够随意关闭。Java 中给我们提供了很多容器类，可以方便地用来构建连接池，如 Vector、Stack 等。在系统初始化时，根据配置创建连接并放置在连接池中，以后所使用的连接都是从该连接池中获取的，这样就可以避免连接随意建立、关闭造成的开销（当然，我们没有办法避免 Java 的 Garbage Collection 带来的开销）。

有了这个连接池，下面就可以提供一套自定义的分配、释放策略。当客户请求数据库连接时，首先看连接池中是否有未分配出去的连接。如果存在空闲连接则把连接分配给客户，并作相应处理。具体处理策略，在关键议题中会详述，主要的处理策略就是标记该连接为已分配。若连接池中没有空闲连接，就在已经分配出去的连接中，寻找一个合适的连接给客户，此时该连接在多个客户间复用。

当客户释放数据库连接时，可以根据该连接是否被复用，进行不同的处理。如果连接没有使用者，就放入到连接池中，而不是被关闭。可以看出，正是这套策略保证了数据库连接的有效复用。

16.5 数据架构规划与设计

16.5.1 数据库设计与类的设计融合

对类和类之间关系的正确识别是数据模型的关键所在。本节将讨论如何发现、识别以及描述类。要想将建模过程缩减为一个简单的、逐步进行的过程是不太可能的。从本质上讲，建模是一项艺术。对一个给定的复杂情况而言，不存在唯一正确的数据模型，然而却存在好的数据模型。一个企业或机构的某个数据模型可能会优于另一个数据模型，但就如何为一个特定的系统建立数据模型，却没有唯一的解决方案。

好模型的目标是将工程项目整个生存期内的花费减至最小，同时也会考虑到随时间

的推移系统将可能发生的变化，因而设计时也要很容易地能适应这些变化。因此，将目光集中在最大限度地降低开发费用上是一个错误。

16.5.2　数据库设计与 XML 设计融合

WWW 的迅速发展，使其成为全球信息传递和共享日益重要和最具潜力的资源，电子商务、电子图书和远程教育等全新领域的需求和发展，使 Web 数据变得更加复杂和多样化，利用传统数据库技术很难存储和管理所有不同的 Web 数据。

目前，XML 正在成为 Internet 上数据描述和交换的标准，并且将来会代替 HTML 而成为 Web 上保存数据的主要格式。

XML 文档分为两类：一类是以数据为中心的文档，这种文档在结构上是规则的，在内容上是同构的，具有较少的混合内容和嵌套层次，人们只关心文档中的数据而并不关心数据元素的存放顺序，这种文档简称为数据文档，它常用来存储和传输 Web 数据。另一类是以文档为中心的文档，这种文档的结构不规则，内容比较零散，具有较多的混合内容，并且元素之间的顺序是有关的，这种文档常用来在网页上发布描述性信息、产品性能介绍和 E-mail 信息等。

Web 上存有大量的 XML 文档，并需要持久保存，这一需求引发了人们对 XML 文档的存储技术研究。已经提出的 XML 文档的存储方式有两种：基于文件的存储方式和数据库存储方式。

（1）基于文件的存储方式。基于文件的存储方式是指将 XML 文档按其原始文本形式存储，主要存储技术包括操作系统文件库、通用文档管理系统和传统数据库的列（作为二进制大对象 BLOB 或字符大对象 CLOB）。这种存储方式需维护某种类型的附加索引，以建立文件之间的层次结构。基于文件的存储方式的特点：无法获取 XML 文档中的结构化数据；通过附加索引可以定位具有某些关键字的 XML 文档，一旦关键字不确定，将很难定位；查询时，只能以原始文档的形式返回，即不能获取文档内部信息；文件管理存在容量大、管理难的缺点。

（2）数据库存储方式。数据库在数据管理方面具有管理方便、存储占用空间小、检索速度快、修改效率高和安全性好等优点。一种比较自然的想法是采用数据库对 XML 文档进行存取和操作，这样可以利用相对成熟的数据库技术处理 XML 文档内部的数据。数据库存储方式的特点：能够管理结构化和半结构化数据；具有管理和控制整个文档集合本身的能力；可以对文档内部的数据进行操作；具有数据库技术的特性，如多用户、并发控制和一致性约束等；管理方便，易于操作。

在某种程度上，XML 及其一系列相关技术就是一个数据库系统。它提供了传统数据库所具有的特点，如存储（以 XML 文档形式）、数据库的模式（DTD 或 XMLSchema）、查询语言（XQuery、XPath、XQL 和 XML-QL 等）和编程接口（如 SAX、DOM）等。但与传统数据库相比，它在存储、索引、安全、多用户访问和事务管理等方面还存在不

足之处。在一定的环境下，例如当数据量和操作用户较少并且性能要求不高的情况下，XML 文档能够作为数据库在应用程序中使用。如果应用程序有许多操作用户，并且要求严格的数据完整性和性能要求，则不宜采用 XML 文档。

XML 数据库是一组 XML 文档的集合，并且是持久的和可操作的；有专门的 DBMS 管理（不是 XML 文件系统）；文档都是有效的（即符合某一模式）；文档的集合可能基于多个模式文件（即文件扩展名为.xsd），多个模式文件之间可能有语法和语义上的相互联系。

16.6 实战案例——电子商务网站（网上商店 PetShop）

PetShop 是一个范例，微软用它来展示.Net 企业系统开发的能力。PetShop 随着版本的不断更新，至现在基于.Net 2.0 的 PetShop4.0 为止，整个设计逐渐变得成熟而优雅，有很多可以借鉴之处。PetShop 是一个小型的项目，系统架构与代码都比较简单，却也凸现了许多颇有价值的设计与开发理念。

1．PetShop 的系统架构设计

PetShop 的表示层是用 ASP.Net 设计的，也就是说，它应是一个 BS 系统。在.Net 中，标准的 BS 分层式结构如图 16-12 所示。

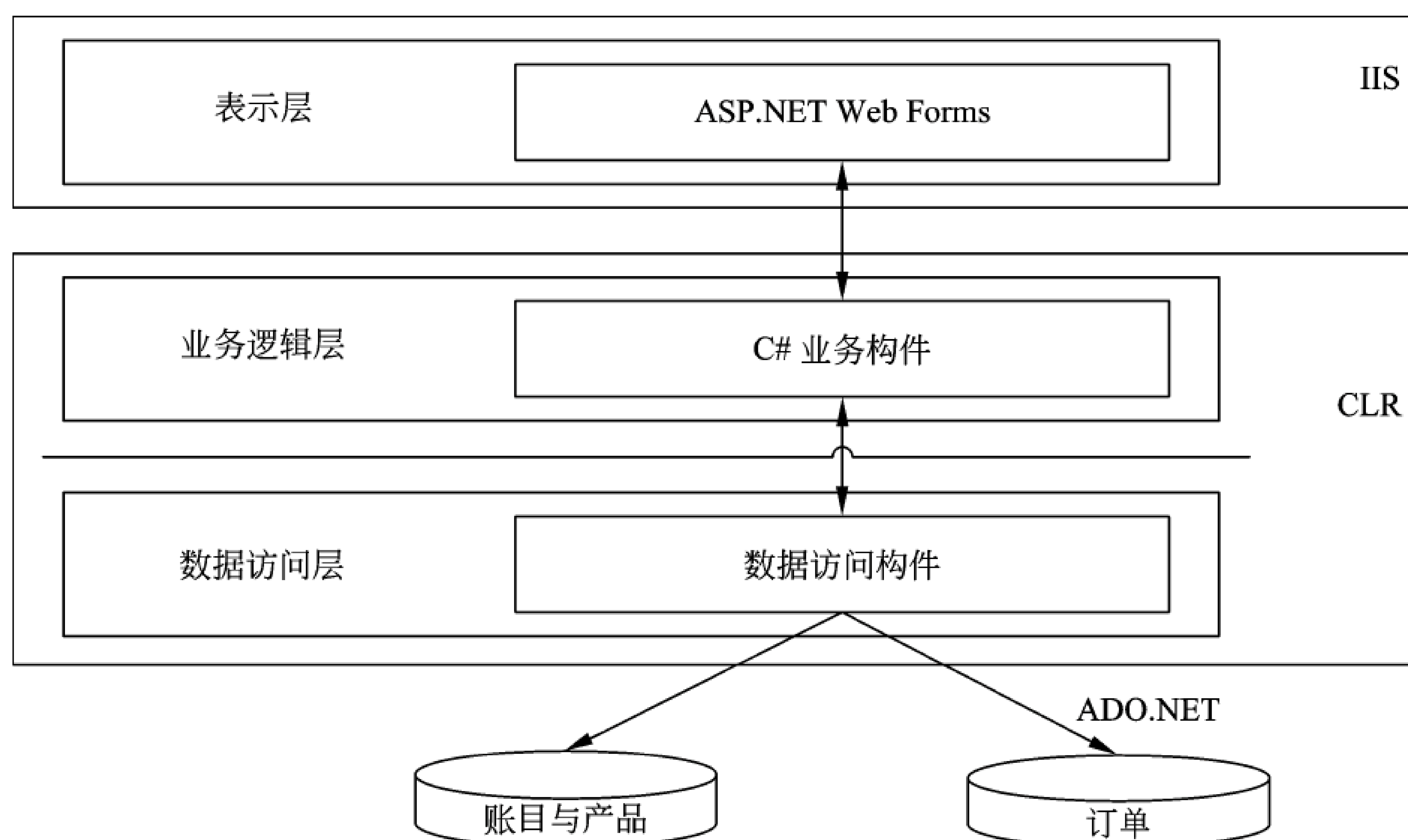


图 16-12　Net 中标准的 BS 分层式结构

随着 PetShop 版本的更新，其分层式结构也在不断的完善，例如 PetShop 2.0，就没有采用标准的三层式结构，如图 16-13 所示。

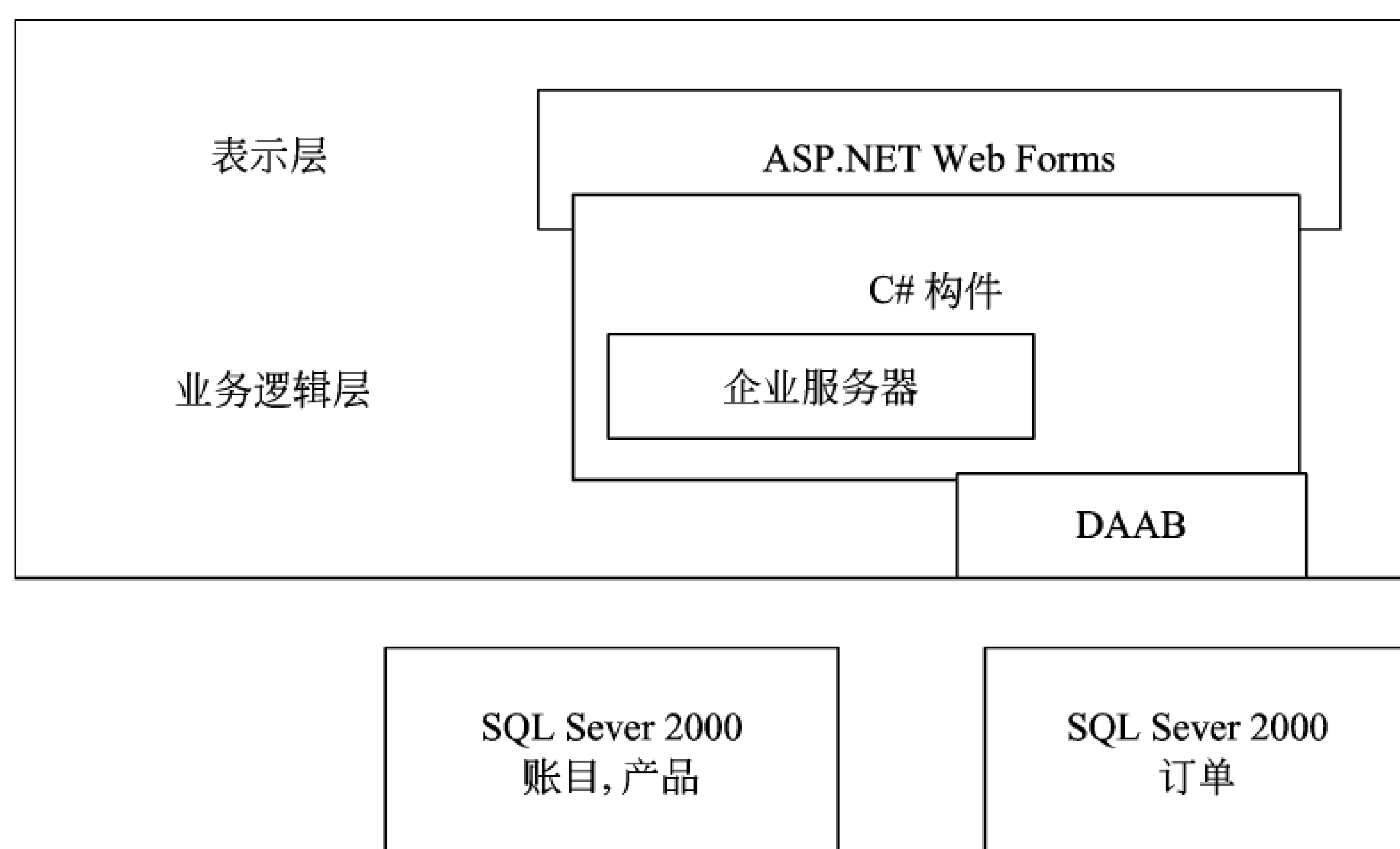


图 16-13　PetShop 2.0 的体系架构

从图 16-13 中可以看到，并没有明显的数据访问层设计。这样的设计虽然提高了数据访问的性能，但也同时导致了业务逻辑层与数据访问的职责混乱。一旦要求支持的数据库发生变化，或者需要修改数据访问的逻辑，由于没有清晰的分层，会导致项目做大的修改。而随着硬件系统性能的提高，以及充分利用缓存、异步处理等机制，分层式结构所带来的性能影响几乎可以忽略不计。

PetShop 3.0 纠正了此前层次不明的问题，将数据访问逻辑作为单独的一层独立出来。PetShop 3.0 的体系架构如图 16-14 所示。

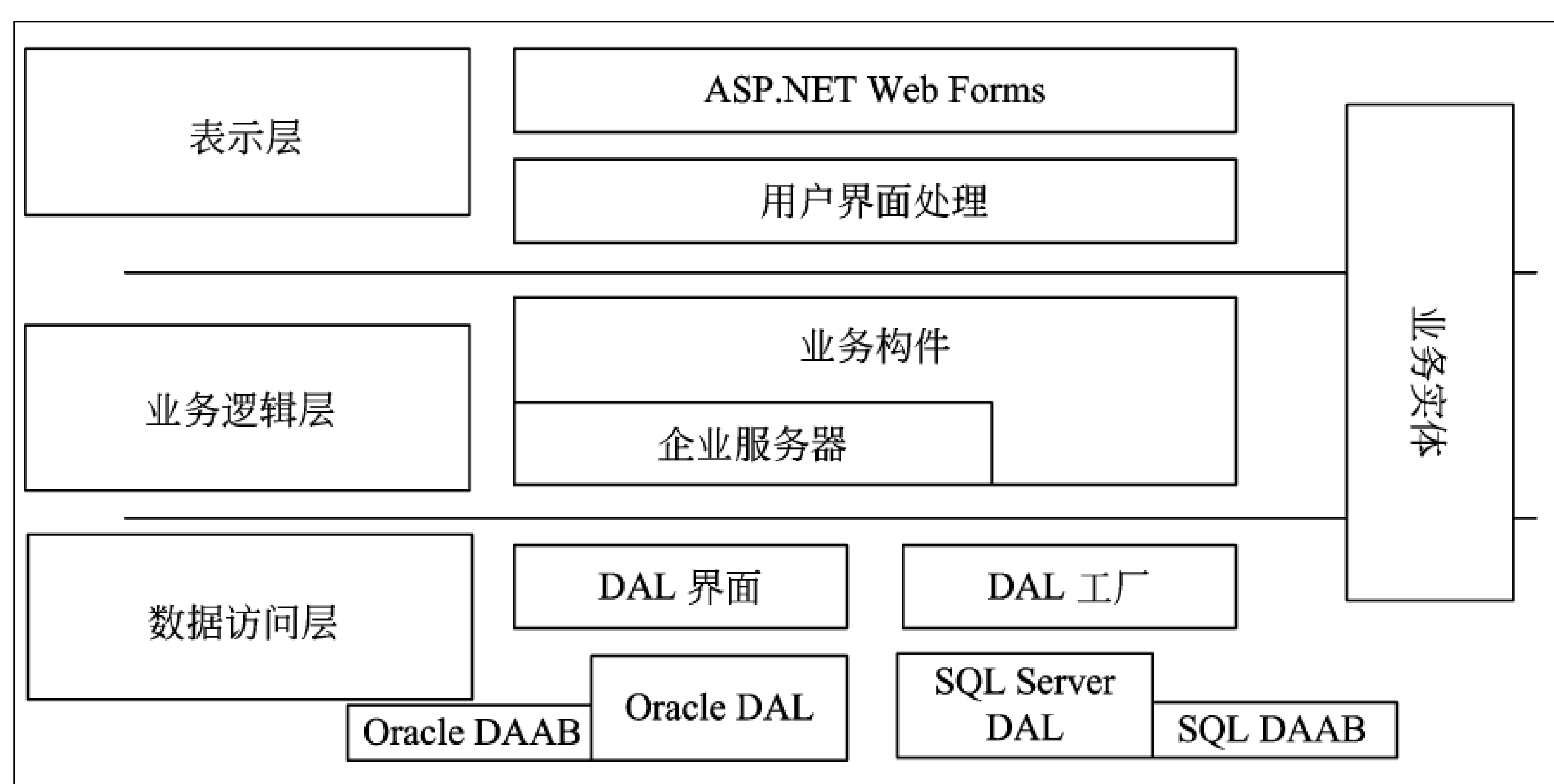


图 16-14　PetShop 3.0 的体系架构

PetShop 4.0 基本上延续了 3.0 的结构，但在性能上作了一定的改进，引入了缓存和异步处理机制，同时又充分利用了 ASP.Net 2.0 的新功能 MemberShip。因此，PetShop 4.0

的系统架构如图 16-15 所示。

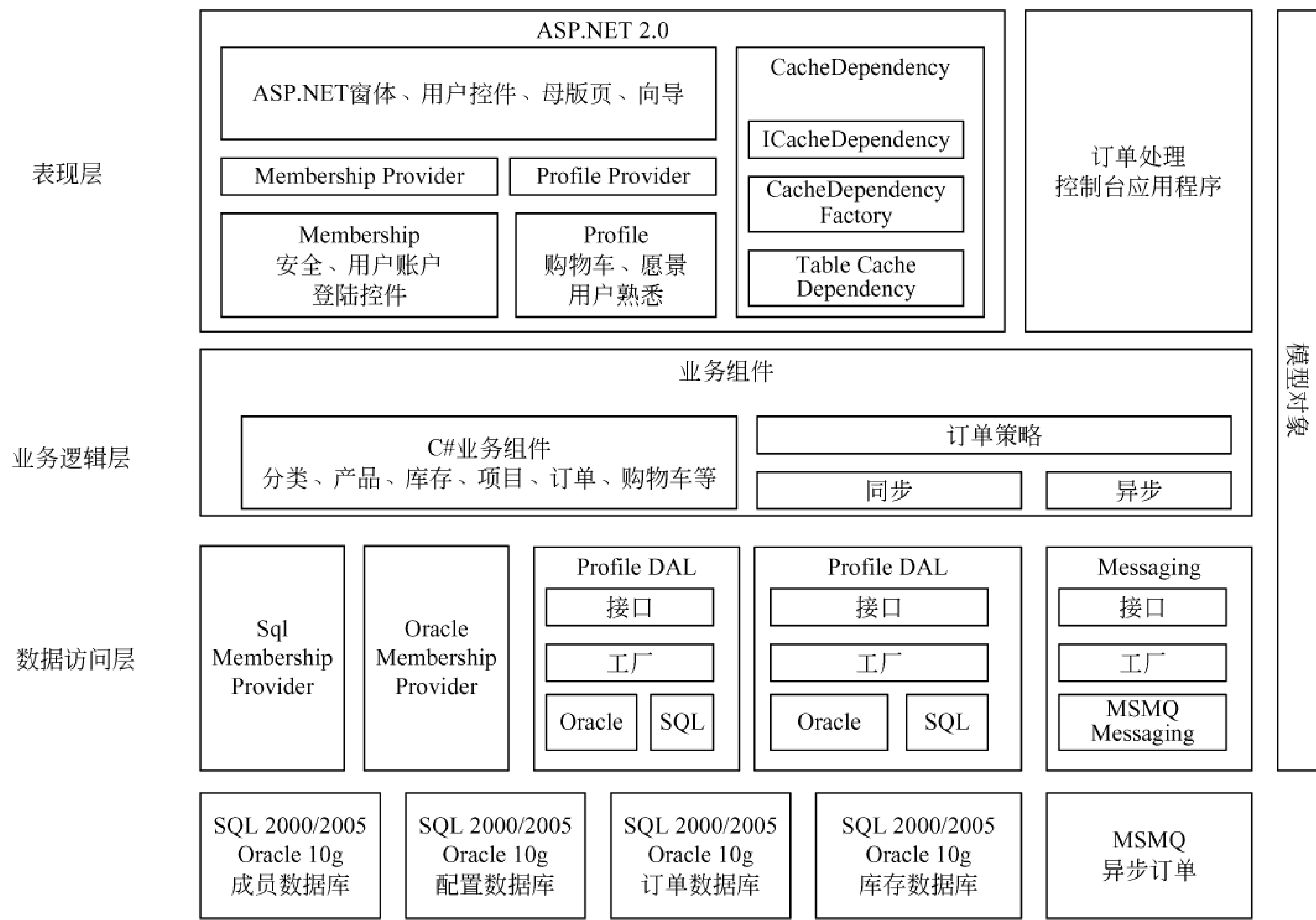


图 16-15 PetShop 4.0 的体系架构

比较 3.0 和 4.0 的系统架构图，其核心的内容并没有发生变化。在数据访问层（DAL）中，仍然采用 DAL Interface 抽象出数据访问逻辑，并以 DAL Factory 作为数据访问层对象的工厂模块。对于 DAL Interface 而言，分别有支持 MS-SQL 的 SQL Server DAL 和支持 Oracle 的 Oracle DAL 具体实现，而 Model 模块则包含了数据实体对象，其详细的模块结构如图 16-16 所示。

可以看到，在数据访问层中，完全采用了“面向接口编程”思想。抽象出来的 IDAL 模块，脱离了与具体数据库的依赖，从而使得整个数据访问层有利于数据库迁移。DALFactory 模块专门管理 DAL 对象的创建，便于业务逻辑层访问。SQLServerDAL 和 OracleDAL 模块均实现 IDAL 模块的接口，其中包含的逻辑就是对数据库的 Select、Insert、Update 和 Delete 操作。因为数据库类型的不同，对数据库的操作也有所不同，代码也会因此有所区别。

此外，抽象出来的 IDAL 模块，除了解除了向下的依赖之外，对于其上的业务逻辑层同样仅存在弱依赖关系，如图 16-17 所示。

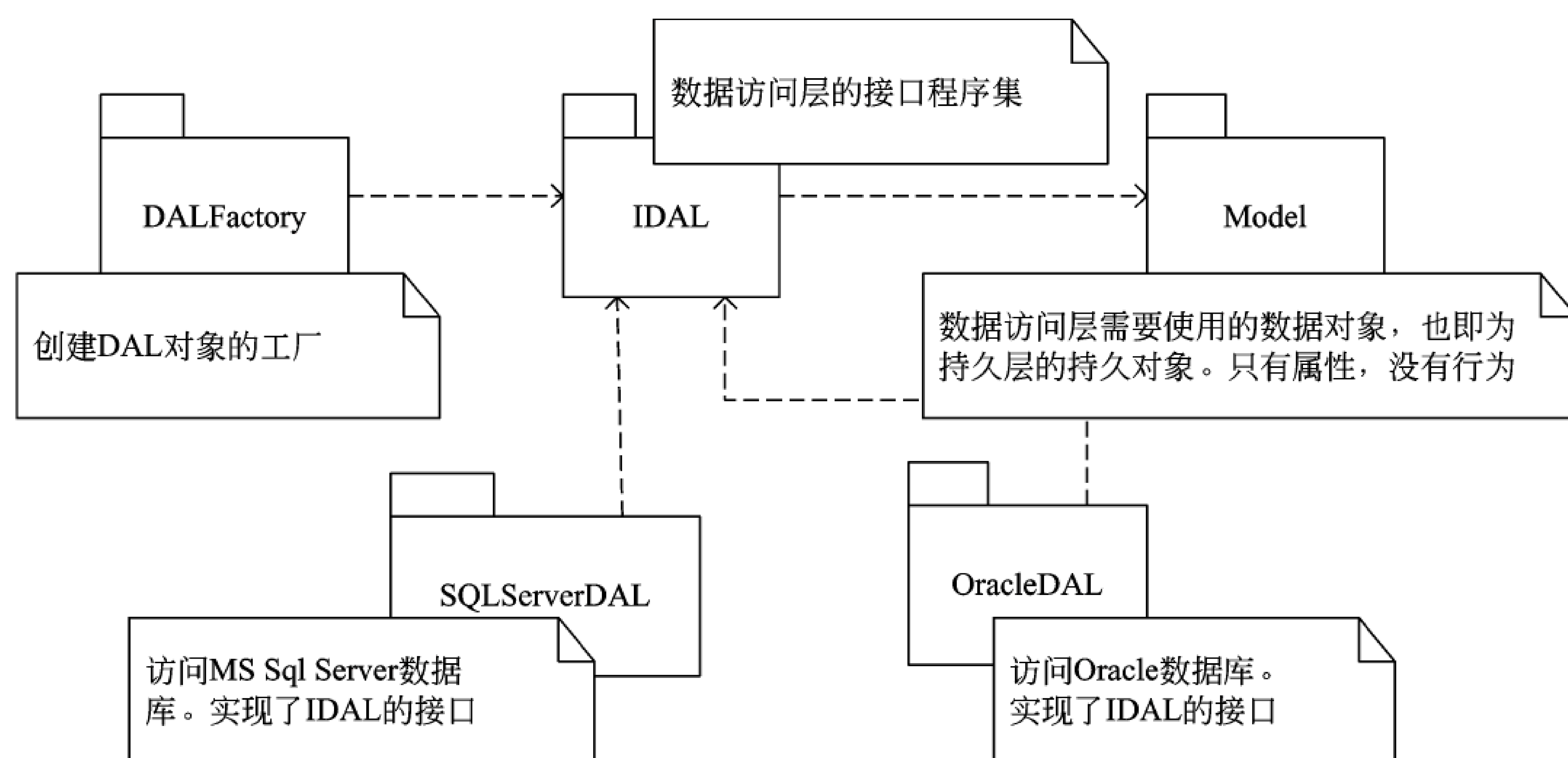


图 16-16　数据访问层的模块结构图

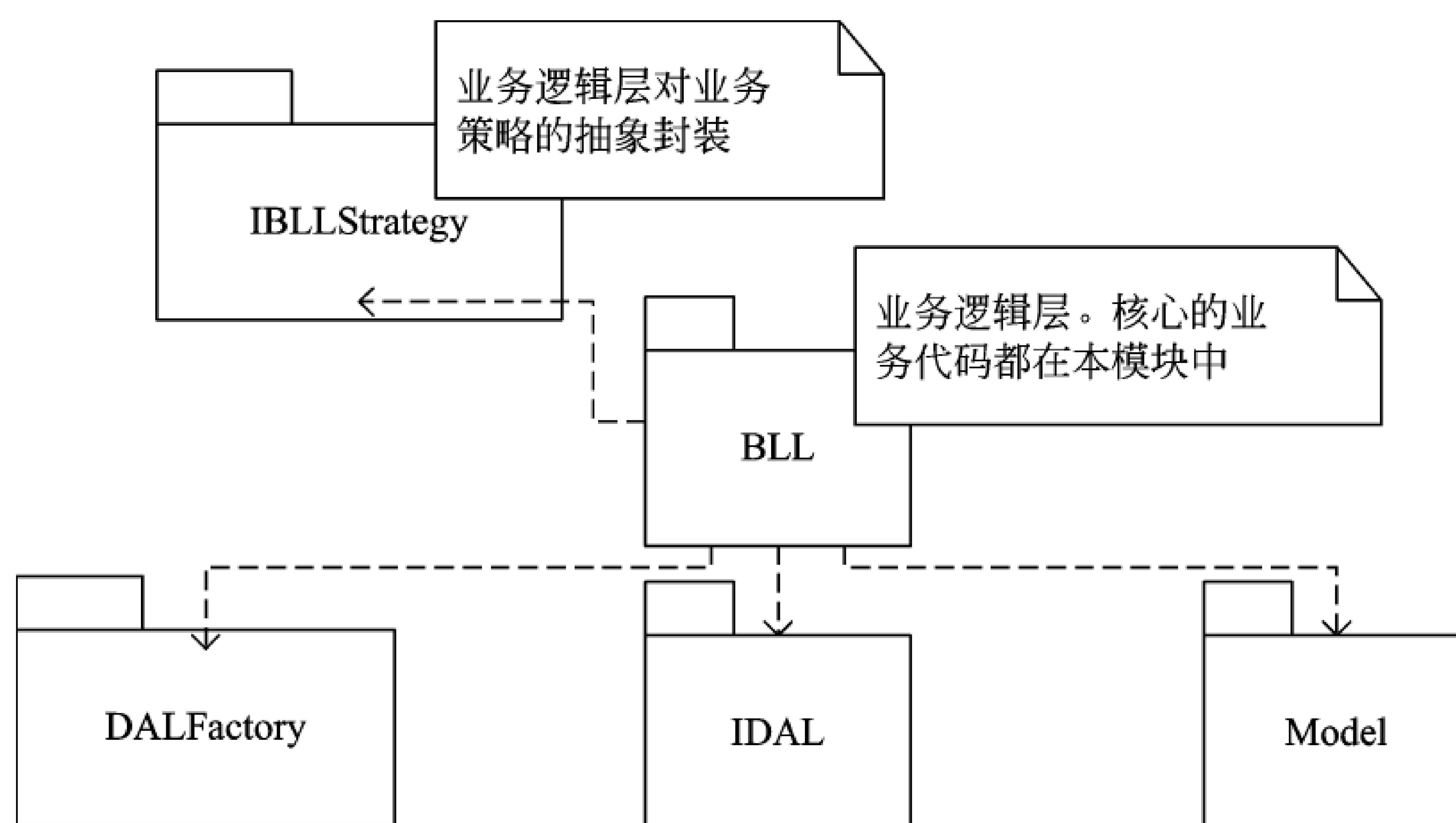


图 16-17　业务逻辑层的模块结构图

图 16-17 中，BLL 是业务逻辑层的核心模块，它包含了整个系统的核心业务。在业务逻辑层中，不能直接访问数据库，而必须通过数据访问层。注意，图 16-17 中对数据访问业务的调用，是通过接口模块 IDAL 来完成的。既然与具体的数据访问逻辑无关，则层与层之间的关系就是松散耦合的。如果此时需要修改数据访问层的具体实现，只要不涉及到 IDAL 的接口定义，那么业务逻辑层就不会受到任何影响。毕竟，具体实现的 SQLServerDAL 和 OracalDAL 根本就与业务逻辑层没有半点关系。

因为在 PetShop 4.0 中引入了异步处理机制，插入订单的策略可以分为同步和异步，两者的插入策略明显不同。但对于调用者而言，插入订单的接口是完全一样的，所以 PetShop 4.0 中设计了 IBLLStrategy 模块。虽然在 IBLLStrategy 模块中，仅仅是简单的

IOrderStategy，但同时也给出了一个范例和信息，那就是在业务逻辑的处理中，如果存在业务操作的多样化或者是今后可能的变化，均应利用抽象的原理、或者使用接口、或者使用抽象类，从而脱离对具体业务的依赖。不过在PetShop中，由于业务逻辑相对简单，这种思想体现得不够明显。也正因为此，PetShop将核心的业务逻辑都放到了一个模块BLL中，并没有将具体的实现和抽象严格地按照模块分开。所以表示层和业务逻辑层之间的调用关系，其耦合度相对较高。

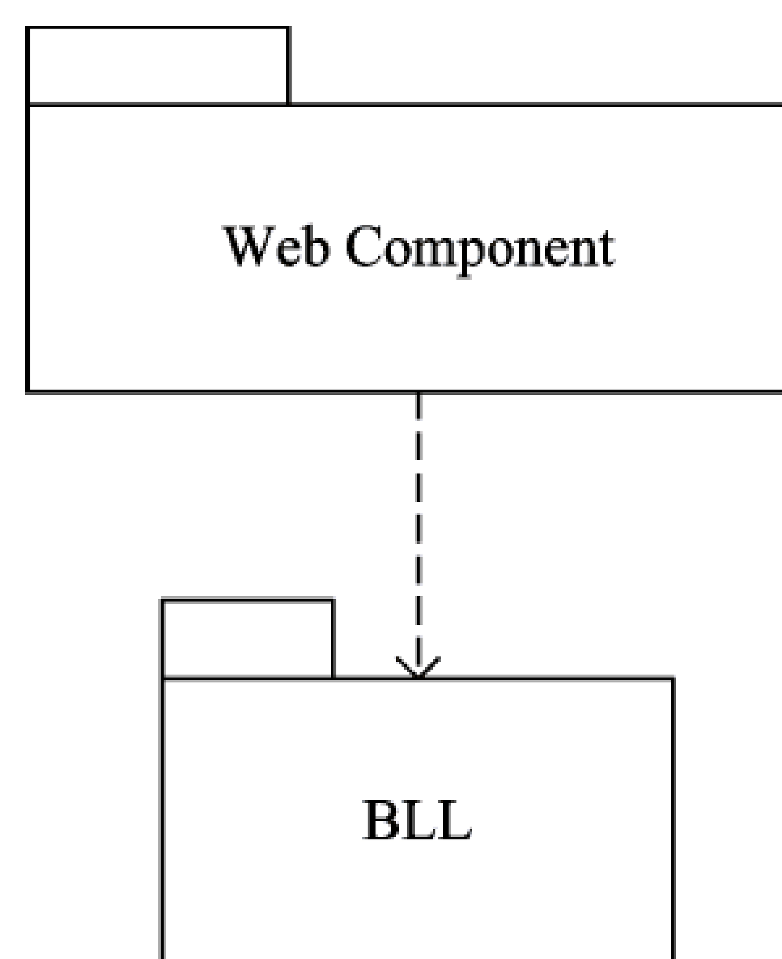


图 16-18 表示层的模块结构

图16-18表示层的模块结构图中，各个层次中还引入了辅助的模块，如数据访问层的Messaging模块，是为异步插入订单的功能提供，采用了MSMQ（Microsoft Messaging Queue）技术，而表示层的CacheDependency则提供缓存功能。

第 17 章　企业集成架构设计

企业信息集成是解决“孤岛”问题的需要，技术发展的同时也推动了集成架构等相关的研究。企业集成平台的核心是企业集成架构，包括信息、过程、应用集成的架构。本章从集成平台概念出发，探讨相关的标准、规范、技术及设计模型，包括面向企业整体集成模型和作用。

17.1　企业集成平台

信息时代的企业集成需要在一个开放的计算机支撑环境下实现。企业集成平台（Enterprise Integration Platform，EIP）技术是近年来用于企业信息系统集成的一种先进的计算机软件技术，其目的是能够根据业务模型的变化快速地进行信息系统的配置和调整，保证不同系统、应用、服务或操作人员之间顺畅地互操作，进而提高企业适应市场变化的能力，使企业能够在复杂多变的市场环境中生存。

企业集成的水平在很大程度上取决于企业内部各种系统、应用或服务的集成化运行水平，良好的软件支持工具可以帮助企业加快实现企业系统集成。作为支持企业集成化运行的工具，企业集成平台的主要功能是为企业中各种数据、系统和过程等多种对象的协同运行提供各种公共服务及运行时的支撑环境，从而降低实现企业内部的信息孤岛集成的复杂度，提高应用间集成的有效性，将信息系统实施规划中确定的企业中各种应用系统、服务、人员、信息资源及数字化设备的协同关系物化到集成化运行的可执行系统中去。

17.1.1　企业集成平台的概念

企业集成平台概念的提出和发展来自于企业应用需求和计算机技术发展两方面的驱动。一方面，企业中各种业务信息系统（包含各种遗留信息系统）数量的增加为企业集成平台产生了需求拉动的作用；另一方面，计算机及软件技术的发展是产生企业集成平台的技术推动力。

实现企业集成的技术和手段多种多样，早期比较简单的集成方式是通过在不同的应用之间开发一对一的专用接口来实现应用之间的数据集成，即采用点到点的集成方式。这种点到点的集成方式的优点是比较直观，在企业应用数量少时易实现。但这种方式也存在比较多的问题：工作量大；集成系统的维护费用高，系统升级和扩展困难；不易于标准化，由于接口数量多，给系统管理造成比较大的困难；一般仅能够解决应用系统之

间的数据集成问题，难以用来支持过程集成和应用之间的协调。

为了克服点到点集成方式给企业应用系统集成和维护管理带来的困难，人们提出了采用集成平台的方式来实现企业集成。企业集成平台是一个支持复杂信息环境下信息系统开发、集成和协同运行的软件支撑环境。它基于各种企业经营业务的信息特征，在异构分布环境（操作系统、网络、数据库）下为应用提供一致的信息访问和交互手段，对其上运行的应用进行管理，为应用提供服务，并支持企业信息环境下各特定领域的应用系统的集成。

经过多年的发展，集成平台已经成为支持企业集成的先进和有效的方法。基于集成平台，可以使分散的信息系统通过一个单一的接口，以可管理、可重复的方式实现单点集成，使企业内的所有应用都可以通过集成平台进行通信和数据交换，实现广义范围内和深层次上的企业资源共享和集成。图 17-1 给出了企业集成平台的示意图。

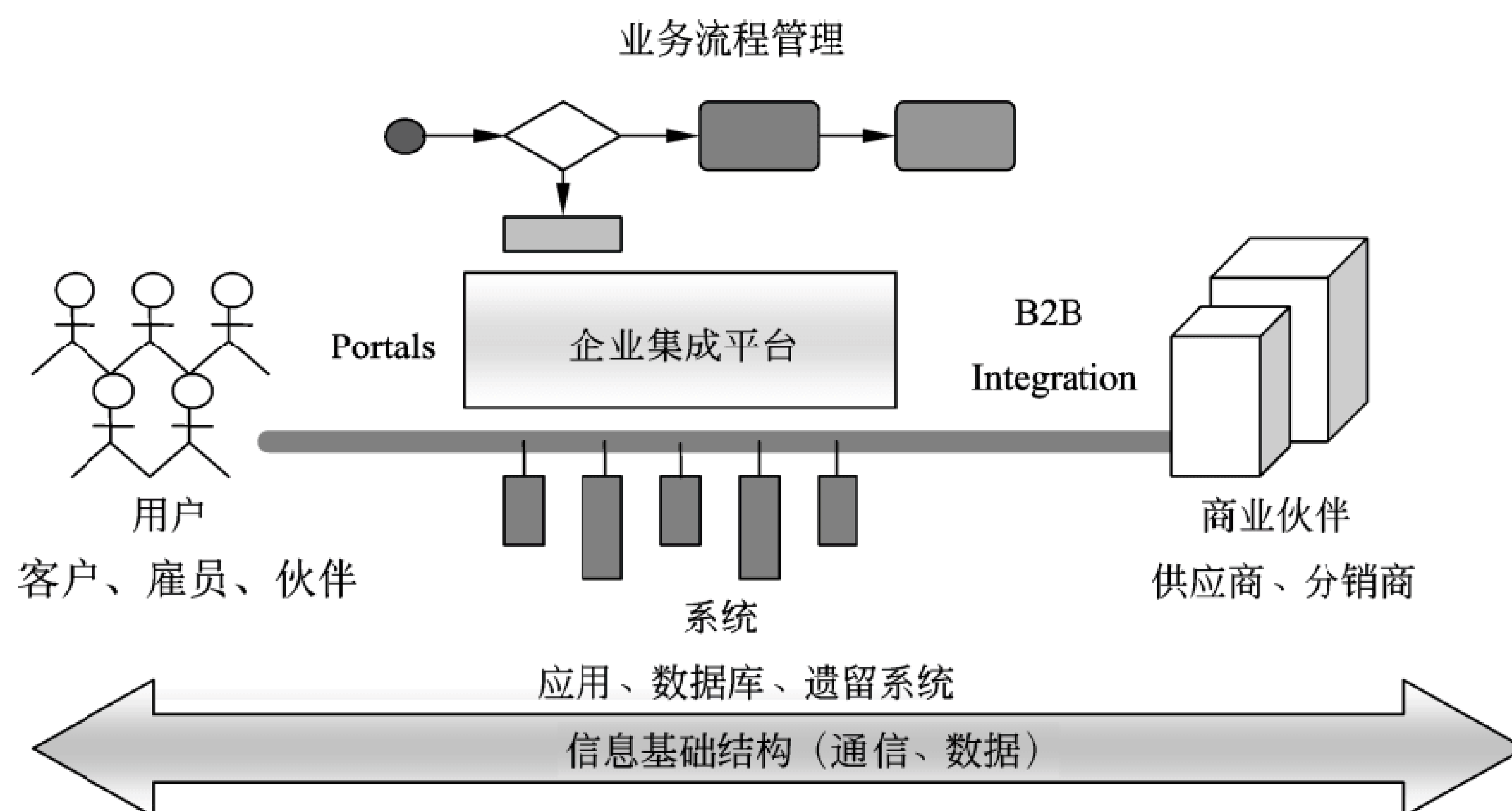


图 17-1　企业集成平台的应用架构

企业集成平台的产生和发展，使得企业应用软件的开发方式较传统方式发生了很大变化，也使得应用系统维护和扩展的难度及费用大为减少。应用集成平台提供的应用软件集成机制和接口可以实现应用间的透明信息交换，使得在异构分布环境下的应用软件通过该接口集成到平台上，共享平台所拥有的资源。采用集成平台可以大大降低集成的复杂度，提高集成的有效性。

由于其诸多的优点，从 20 世纪 80 年代中期以后，集成平台的概念和产品在全世界范围内得到了广泛的推广应用，出现了狭义的集成平台和广义的集成平台两种概念。狭义的集成平台是指一个软件平台，它为企业内多个应用软件系统或组件间的信息共享与互操作提供所需的通用服务，达到降低企业内（间）多个应用软件系统或系统之间的集成复杂性的目的。广义的集成平台则是指由支撑软件系统（狭义集成平台）同其他完成不同业务的逻辑功能的各应用系统一起组成数字化企业的协同运行环境。但无论是广义

的集成平台，还是狭义的集成平台，其核心的内容都是为企业提供集成所需要的服务，并对集成系统进行管理。

集成平台是支持企业集成的支撑环境，包括硬件、软件、软件工具和系统，通过集成各种企业应用软件形成企业集成系统。由于硬件环境和应用软件的多样性，企业信息系统的功能和环境都非常复杂，因此，为了能够较好地满足企业的应用需求，作为企业集成系统支持环境的集成平台，其基本功能主要如下。

1）通信服务

提供分布环境下透明的同步 / 异步通信服务功能，使用户和应用程序无需关心具体的操作系统和应用程序所处的网络物理位置，而以透明的函数调用或对象服务方式完成它们所需的通信服务要求。

2）信息集成服务

为应用提供透明的信息访问服务，通过实现异种数据库系统之间数据的交换、互操作、分布数据管理和共享信息模型定义（或共享信息数据库的建立），使集成平台上运行的应用、服务或用户端能够以一致的语义和接口实现对数据（数据库、数据文件、应用交互信息）的访问与控制。

3）应用集成服务

通过高层应用编程接口来实现对相应应用程序的访问，这些高层应用编程接口包含在不同的适配器或代理中，被用来连接不同的应用程序。这些接口以函数或对象服务的方式向平台的组件模型提供信息，使用户在无需对原有系统进行修改（不会影响原有系统的功能）的情况下，只要在原有系统的基础上加上相应的访问接口就可以将现有的、用不同的技术实现的系统互联起来，通过为应用提供数据交换和访问操作，使各种不同的系统能够相互协作。

4）二次开发工具

是集成平台提供的一组帮助用户开发特定应用程序（如实现数据转换的适配器或应用封装服务等）的支持工具，其目的是简化用户在企业集成平台实施过程中（特定应用程序接口）的开发工作。

5）平台运行管理工具

是企业集成平台的运行管理和控制模块，负责企业集成平台系统的静态和动态配置、集成平台应用运行管理和维护、事件管理和出错管理等。通过命名服务、目录服务、平台的动态静态配置，以及其中的关键数据的定期备份等功能来维护整个服务平台的系统配置及稳定运行。

17.1.2　集成平台的标准化

集成平台上集成的应用软件系统通常都是由不同的软件厂家提供的产品，具有很强的异构性，所以在集成平台中需要广泛采用新的开放性标准。研究和发展系统集成的相

关标准，不断地使平台的接口和服务标准化，可以显著提高集成平台系统的适应性和可扩展性，减少异构性给集成带来的障碍。采用标准化的技术也是提高集成平台系统开放性和软件模块可重用性的重要方法。

集成平台的标准化内容涉及通信协议、中间件、企业建模、工作流管理系统、Internet 环境下的数据交换、产品数据标准和应用系统集成的标准等。Goldstone 技术公司在国际标准化组织定义的开放系统互联（ISO/OSI）的 7 层网络应用模型的基础上，给出了图 17-2 所示的集成平台的 12 层 OSI 模型。

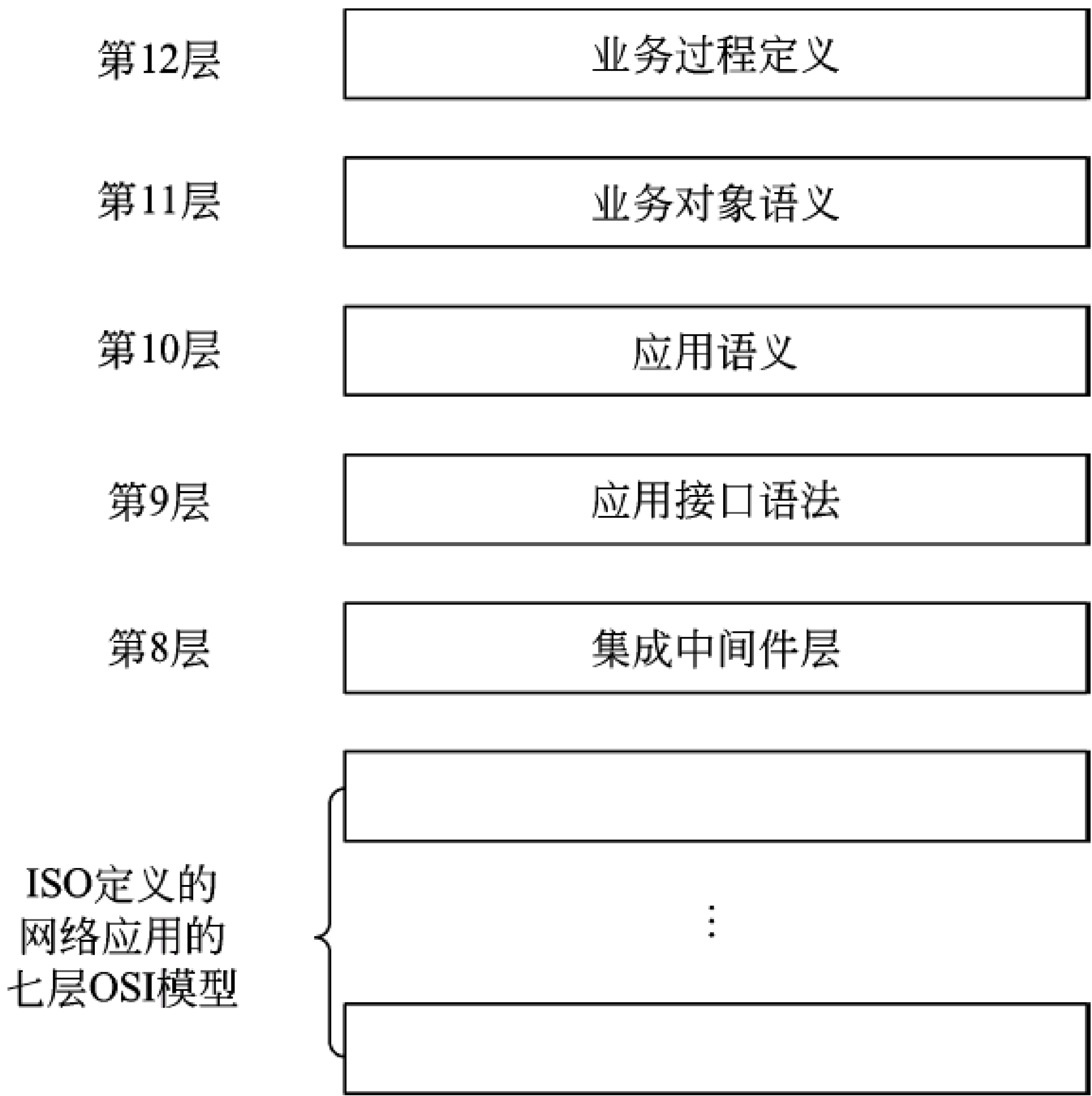


图 17-2　企业集成平台的 12 层 OSI 模型

在这个 12 层 OSI 模型中，下面的 7 层依然是采用 ISO 关于网络应用 7 个层次的定义。第 8 层为支持应用集成的中间件层，它为集成平台提供商实施企业系统集成提供了可扩展集成的架构。第 9 层为应用开发商定义的应用间方法（服务）调用、接收/发送消息格式的接口语法层。第 10 层为应用提供商和集成平台提供商共同提供的用来描述应用软件系统结构和内涵的应用语义层。第 11 层作为业务语义描述层，供业务操作人员和信息管理人员用来定义基于模型操作的业务对象的数据结构及其语义。第 12 层为业务过程层，用来为业务操作人员定义企业关键业务流程及流程之间的交互关系。

17.1.3　实现技术的发展趋势

通过分析国内外集成平台的应用及发展情况，结合企业集成系统对集成平台实施提出的要求和计算机软件技术的发展趋势，企业集成技术有如下的发展趋势。

1．集成的技术实现从 2 层到 n 层过渡

传统的集成实现一般采用图 17-3 所示的两层 C/S 或 B/S 结构，这样的系统将业务逻辑和应用表示逻辑封装在一起。这个封装在一起的逻辑模块可以安装在客户端应用上，也可以安装在服务器上，但是无论是在服务器端，还是在客户端，由于业务逻辑和应用表示逻辑的紧密捆绑，对系统的升级和扩展都带来了比较大的困难。

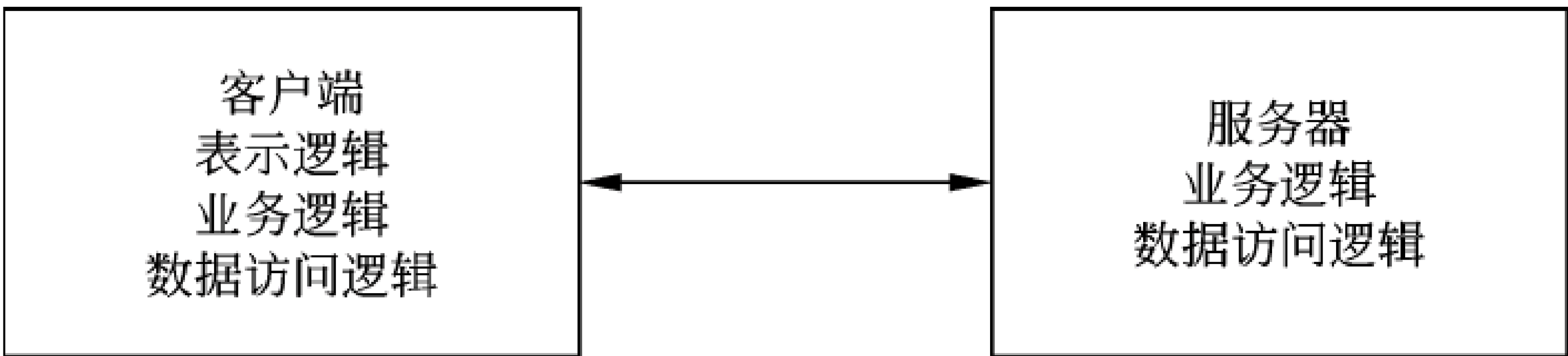


图 17-3　集成技术的两层实现

未来的集成平台将采用图 17-4 所示的 n 层系统集成方式，将业务过程逻辑、业务表示逻辑等进行分离，将每层的功能集中在一个特定的角色上，这样可以得到一个非常便于进行系统功能扩展、逻辑修改的应用集成框架，进而提高集成平台和集成系统的柔性。

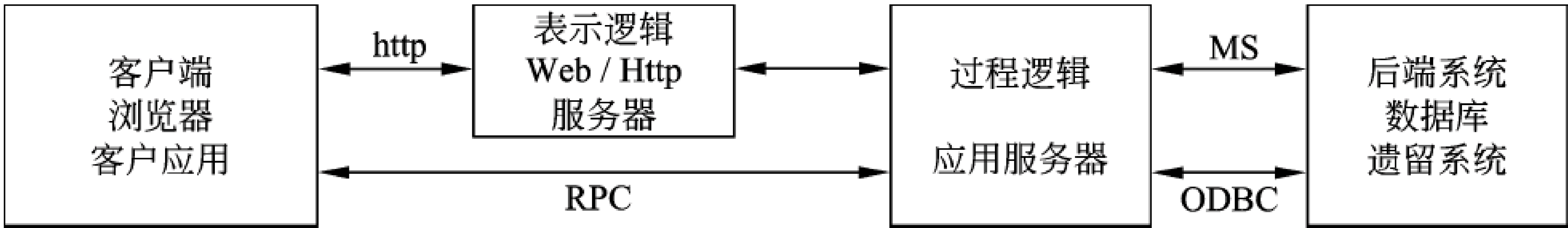


图 17-4　集成技术的 n 层实现

2．集成支持的方式从面向信息集成扩充到面向过程集成、服务集成

面向信息的集成主要是针对设计、制造和管理部门中大量存在的自动化孤岛和信息孤岛而提出来的，其目的是为了解决企业内不同应用和系统间的数据共享和集成。这些应用系统分布在网络环境下的异构计算机系统中，它们所管理和操作的数据格式和存储方式各异，实现信息集成就是要实现数据的转换（不同数据格式和存储方式之间的转换）、数据源的统一（同一个数据仅有一个数据入口）、数据一致性的维护、异构环境下不同的应用系统之间的数据传送。面向信息的集成主要应用于企业内的数据库和数据源上，其具体的实现方法主要有数据复制、数据捆绑和基于接口的信息集成三种方式。

（1）面向过程的集成（这里主要是指技术层面的过程集成）：通过工作流引擎对企业内业务流程模型的执行来实现业务应用数据或信息在不同应用、子过程或执行任务的人员之间流动（如图 17-5 所示）。采用工作流管理方式可以对业务过程逻辑和应用逻辑

进行分离，实现过程建模和数据、功能的分离，从而可以在保持具体功能单元不变的情况下，通过修改过程模型来改变系统功能，进而提高系统的柔性。面向过程集成需要在信息集成的基础上进行，或者说面向过程集成可能会对信息集成提出新的要求，因为在执行过程模型时，过程模型中包含的各种活动之间（特别是自动应用之间）同样需要信息共享与集成。过程集成更重要的是一种策略行为，它还具有过程逻辑可视化、业务执行过程自动化、业务过程执行状态和性能的实时监控等功能。

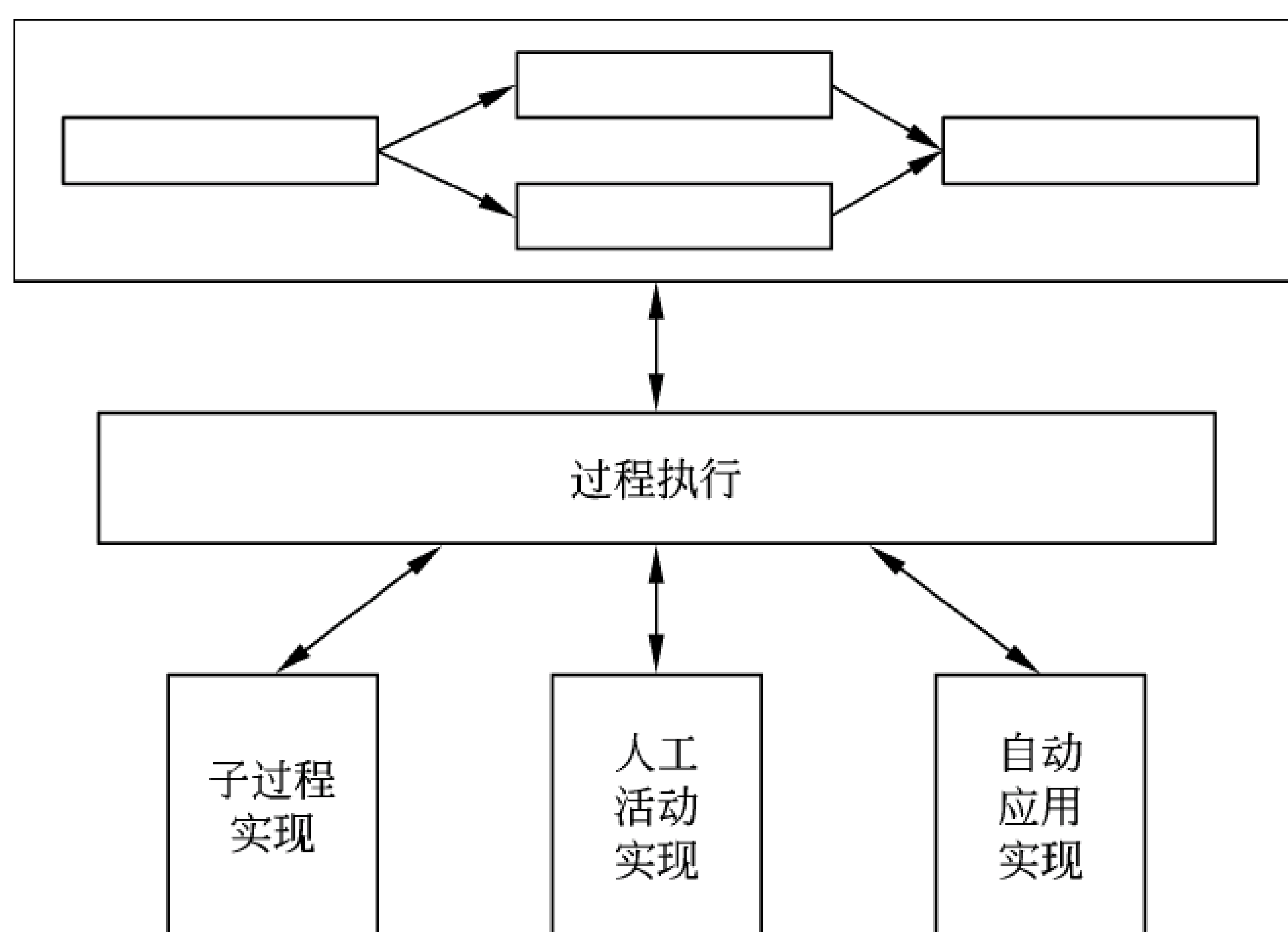


图 17-5 面向过程集成

（2）面向服务的集成（如图 17-6 所示）：主要是为支持大范围内的公共业务过程集成而提出的一种动态集成方式（如供应链企业群体内），可以较好地实现（企业间）具有松散耦合关系的不同应用间的互操作。在这种集成方式中，服务提供者（平台、企业）将应用作为服务部署在 Web 上，通过使用 Web 服务描述语言来描述 Web 服务提供的功能，并通过统一的服务发布与发现协议（Universal Description，Discovery and Integration，UDDI）将其注册到 UDDI 中心。服务请求者使用 UDDI 协议定义的 API 向 UDDI 中心提出服务请求，UDDI 为其寻求到它所需要的服务，并由 UDDI 中心返回服务请求，同时与特定服务进行绑定，在此基础上，服务请求者继而通过 SOAP 协议完成应用服务的调用。基于服务的集成方式对于集成企业原有的系统同样十分方便，在不需要对原有系统进行修改的情况下，只要在原有系统的基础上增加一个对它们进行访问的 SOAP 接口，就可以完成原有系统到集成平台的集成。面向服务的集成将以前主要在企业内部网络基础上实施的集成扩展到了面向开放网络环境下的集成，从而大大扩展了集成的范围。基于服务的集成方式具有最好的柔性和开放性，然而，这种松散的动态集成方式牺牲了性能和网络流量。

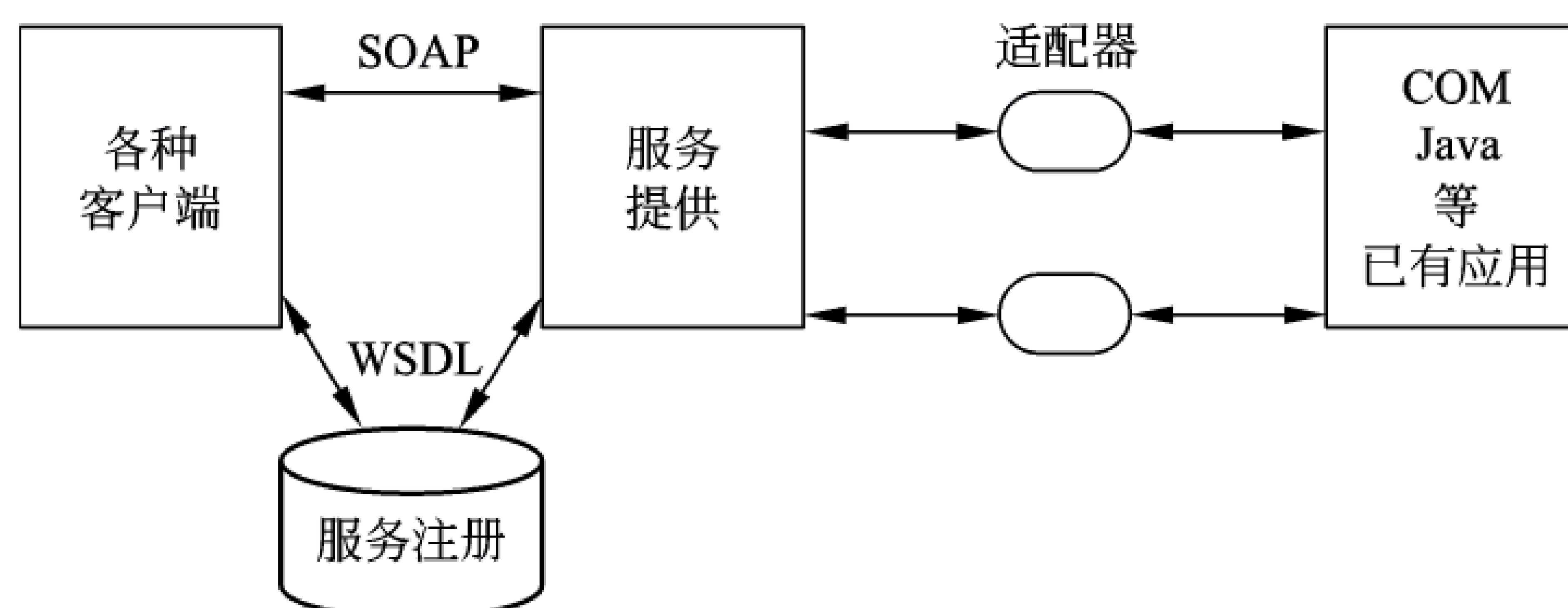


图 17-6　面向服务集成

3．集成规范的标准化程度不断提高

开放性和标准化在集成的实现技术中的重要性已经得到广泛的认同。从数据描述的角度来看，数据结构的定义已经由原来的各个应用专有数据类型、行业内的标准数据表达（如 STEP、EDI 等），逐渐过渡到具有自描述功能的基于 XML 语言的数据表达与存储。从应用间集成接口的实现与接口表现形式来看，已经从最初的自定义应用编程接口、基于 IDL 接口定义（如 CORBA 或 COM 的接口描述语言），发展到更通用的基于 XML 语言的 Web 服务接口定义语言（WSDL）的集成接口描述。从业务过程定义方面来看，则由不同产品给出的自定义业务过程描述方式，工作流联盟为实现不同工作流产品间互操作而提出的工作流过程定义语言（WPDL），到近来出现的关于如何利用 Web 服务集成架构实现过程集成的基于 XML 语言的商业流程模型描述语言（如 WSFL、BPEL 等）。标准化技术的采用增强了集成平台的开放性和通用性，从而为企业集成提供了更强有力的技术支持。

4．所支持的集成耦合度及集成的粒度的变化

随着编程技术的发展，集成平台所采用的集成实现形式也在不断发展，应用集成的耦合度（松散集成、紧密集成）不断降低，集成范围不断扩大，而集成粒度（对象、组件、服务）也在不断缩小，图 17-7 给出了集成的范围和集成耦合度的对应关系。

随着集成范围的不断扩大，集成的耦合度不断降低。集成耦合度最高的对象间集成方式比较适合于功能单元之间的集成，集成耦合度最低的服务集成方式则能够比较好地实现企业间的集成，集成耦合度中等的组件集成方式可以较好地完成企业内的集成。对象间集成主要通过程序代码级对象之间的调用来实现。组件之间的集成方式则主要通过构建企业内分布式计算环境、采用远程过程调用来实现跨语言、进程和计算机间的基于组件的集成。基于服务的集成方式包括基于消息中间件服务和基于 Web 服务两种。基于消息中间件的服务集成通过消息中间件（如 MSMQ）来实现应用或系统之间的互操作，基于 Web 服务的集成通过 SOAP 消息交换协议（防火墙透明的）来实现 Internet 环境下的分布式计算。由于 Web 服务的方式具有良好的松散耦合集成结构，因此它更适合于用来支持企业间应用的集成。

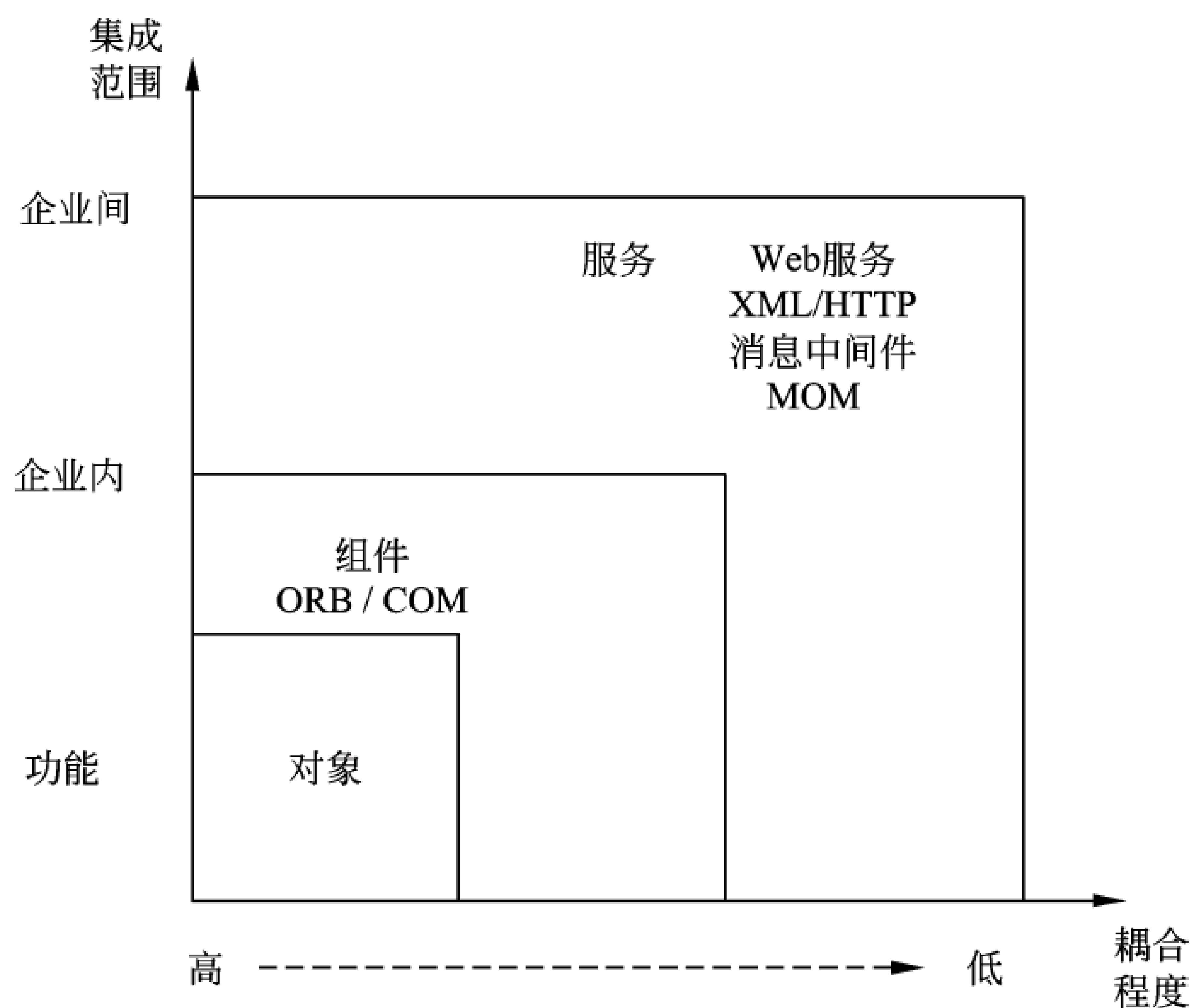


图 17-7 集成尺度与范围、耦合度的关系

17.1.4 集成平台的发展趋势

企业集成平台技术已经逐步成熟，国外已经出现了许多商用产品。从功能上可以将其划分为企业应用集成和业务到业务的集成（B2B）两种。其中，EAI 主要侧重于企业内部的纵向集成，B2B 侧重于支持企业间业务往来的横向集成。目前在市场上的产品主要有 Active Enterprise 3.0（Tibco 公司）、Mercator(Mercator 公司)、MQ Series Integrator（IBM 公司）、WebMethods Enterprise（WebMethods 公司）和 BusinessWare。

1．Active Enterprise 3.0（Tibco 公司）

Tibco 公司主要为具有基本信息技术应用知识的企业用户提供端到端的异构信息系统集成方案。其产品 Active Enterprise 3.0 采用了模块化的结构，它的每一个组件（如数据仓库、集成服务器、消息代理以及监控工具等）都可以在不同机器上独立运行，组件间的通信通过一个连接所有组件的信息总线实现。信息总线采用其独有的串行 UDP 技术实现，能够保证在发生系统级事件（包括通信错误）时及时向相关组件发送通知。考虑到在企业有大量的系统需要集成的情况下，对于整个集成系统的管理和监控将会很复杂，Tibco 采用一个轻型代理实现对整个平台所有层次上的系统和过程进行全面的监控。轻型代理采用广播的形式向信息总线上的所有组件发送必要的监控和管理消息，采用这种方式可以将监控系统对平台系统性能造成的影响降到最低。通过轻型代理和信息总线的协同作用，可以使 Active Enterprise 对新接入的应用或服务具有动态发现能力。另外，Active Enterprise 利用工作流技术为用户提供了强大的业务过程管理能力，用户可以在简

单、直观的过程建模工具的支持下，建立相应的业务过程模型，并通过其工作流引擎同时支持自动化过程和人工型工作流的执行。

2．Mercator（Mercator 公司）

Mercator 由 Enterprise Broker、Web Broker 和 Commerce Broker 三个独立的产品构成。其中 Enterprise Broker 用于企业内应用的集成。Web Broker 是 B2 以外，配合以上产品提供了企业间的流程设计的 GUI 工具 Integration Flow Designer。目前，全世界已经有 5000 套 Mercator 投入运行，Mercator 在集成 SAP R/3 用户方面具有很强的优势。

3．MQ Series Integrator（IBM 公司）

MQ Series Integrator 由消息中间件 MQSeries、消息代理 Integrator 以及实现业务流程自动化的 MQSeries Workflow 构成。MQSeries 是 IBM 开发和销售的消息中间件产品，是消息中间件事实上的标准，它支持 35 种以上的协议，可以用统一的 API 进行异构机种间的连接，主要是进行异步消息处理，但也可以实现实时消息的连接。MQSeries 符合 JMS 标准，可以很容易地与 WebLogic 和 WebSphere 等应用服务器实现连接。MQSeries Workflow 可以采用图形的方式方便地定义跨不同企业系统间的业务流程，也可以对工作流的实例状态进行控制和调整。特别是 MQSeries Workflow 可以将规则嵌入到流程节点中来，这点得到了用户广泛的好评。目前 IBM 已将 WebSphere B2B Integrator 加到这些产品中，以提供包含企业内和企业间集成的综合解决方案。

4．WebMethods Enterprise（WebMethods 公司）

该产品是 WebMethods 公司面向技术型用户提供的 B2B 解决方案，其核心部件是 Active Works。Active Works 提供了通信协议转换、队列管理和队列分配、60 多种适配器、业务流程的控制，XML 变换和 Web 应用接口等 EAI 的基本功能。需要指出的是，WebMethods Enterprise 实现了 EAI 功能的一体化，即各种 EAI 功能可以在一个界面上统一进行设计和操作，所以容易进行应用系统开发和实施。特别地，每一次定义的业务流程都可以以模板的方式进行保存。WebMethods Enterprise 采用总线型体系结构，利用 Java 框架来实现客户适配器的开发，因此具有良好的可扩展性和可用性，特别适用于大型企业的系统集成。WebMethods Enterprise 消息交换的可靠性也较高，它包含一个异步事务协调引擎，并带有一个可用于 MQSeries 的适配器。监控代理、适配器及其各自的流程能够在死机的情况下自动恢复，也可以进行事件的自动重送。在对可靠性有更高要求的应用情况下，还可以在系统的外部配置作业控制器的相关服务模块。在收购了 Active（EAI）和 IntelliFrame（工作流管理系统）后，WebMethods 公司具有为企业提供全面的端到端的集成方案的能力，它为许多主流的 ERP、CRM、基于消息的中间件系统提供了广泛可用的内置适配器。一旦在技术上实现与这些并购产品的全面集成，WebMethods 的领先地位将从目前的 B2B 领域扩展到 EAI 领域。

5．BusinessWare（Vitria Technology 公司）

BusinessWare 产品主要面向技术型用户，它采用以过程为核心的方式实现系统集成。

BusinessWare 产品具体由业务流程管理工具、可作为系统连接的 EAI 平台、实时监视工作流状态的实时分析工具、在应用层担当 B2B 集成的功能模块 4 个部分构成。它的业务流程管理工具具有友好的用户界面，用户可以在不需要事先编程和配置连接器的条件下，进行业务过程的可视化设计。BusinessWare 以 CORBA 技术为核心，采用通道/Hub（集线器）式的系统体系结构，与各系统连接的连接器控制各通道的输入输出，使用连接器开发工具包来支持客户化适配器的开发。BusinessWare 中可以用多个类来定义发布 / 订阅通道的消息，通道采用与域名服务运行方式类似的联邦式分散结构，以实现对不同企业间集成化业务运作提供高性能和高可靠性的服务。在实时分析工具的界面上可以监视所设计流程的运行状态和性能。BusinessWare 可以为那些希望自己进行业务过程建模的企业用户提供标准化的集成服务。

集成平台产品的发展具有以下的主要趋势。

（1）与商用工作流产品的融合发展。

集成平台产品通过与商用工作流产品的融合，一方面将基于工作流的业务流程分析、优化及过程管理功能引入到平台中来，并增强支持业务过程的自动执行能力及平台的可实施性；另一方面，利用商用工作流系统与用户的友好交互能力将人的因素集成到自动执行的企业业务操作过程中来，从而提高系统的柔性与可用性。

（2）与底层集成服务器产品的融合发展。

集成平台产品通过与底层集成服务器产品的融合，一方面可以增加集成平台产品内部各组件模块的无缝集成性，进而提高集成到平台上各应用系统间的互操作能力；另一方面，利用商用构件对企业用户提供从底层服务支撑技术到上层应用、过程集成的一体化支持，以保证集成平台的成功实施。

（3）兼容点到点（Point-to-Point）集成和端到端（End-to-End）集成。

集成平台厂商通过将其传统产品支持的点到点集成（主要指同步集成）方式扩展到端到端集成（侧重于异步集成）方式，以分别适用于企业内部集成所需要的大流量数据交换模式和企业间协同所需要的灵活的小流量数据交换模式。

（4）基于模型的集成与协调。

通过采用统一定义和表示的模型（在一些协议或规则的辅助下实现模型的构造和控制）实现不同应用系统之间的协同工作（应用软件通过模型操作接口实现对模型中定义的产品、过程、资源数据的访问，从而实现不同应用软件之间的无缝集成），这样就可以通过模型在整个生命周期的不断演化来实现企业集成信息系统的演化。

17.2 企业集成平台的实现

17.2.1 数据集成

构建企业集成平台的首要目的是实现数据集成，即为平台上运行的各种应用、系统

或服务，提供具有完整性、一致性和安全性的数据访问、信息查询及决策支持服务。数据集成主要为了解决不同应用和系统间的数据共享和交换需求，具体包括共享信息管理、共享模型管理和数据操作管理三个部分。其中，共享信息管理通过定义统一的集成服务模型和共享信息访问机制，完成对集成平台运行过程中产生数据信息的共享、分发和存储管理；共享模型管理则提供数据资源配置管理、集成资源关系管理、资源运行生命周期管理及相应的业务数据协同监控管理等功能；数据操作管理则为集成平台用户提供数据操作服务，包括多通道的异构模型之间的数据转换、数据映射、数据传递和数据操作等功能服务。

企业运行的业务应用系统采用的体系结构与其实现技术的标准化（规范化）程度，对数据集成的水平有非常大的影响。企业现有各种应用系统的规范化程度不高是影响企业数据集成水平的主要问题，因此，采用先进的软件体系结构和规范化的实现技术是实现良好的数据集成的基础。

企业集成技术架构层次如图 17-8 所示。

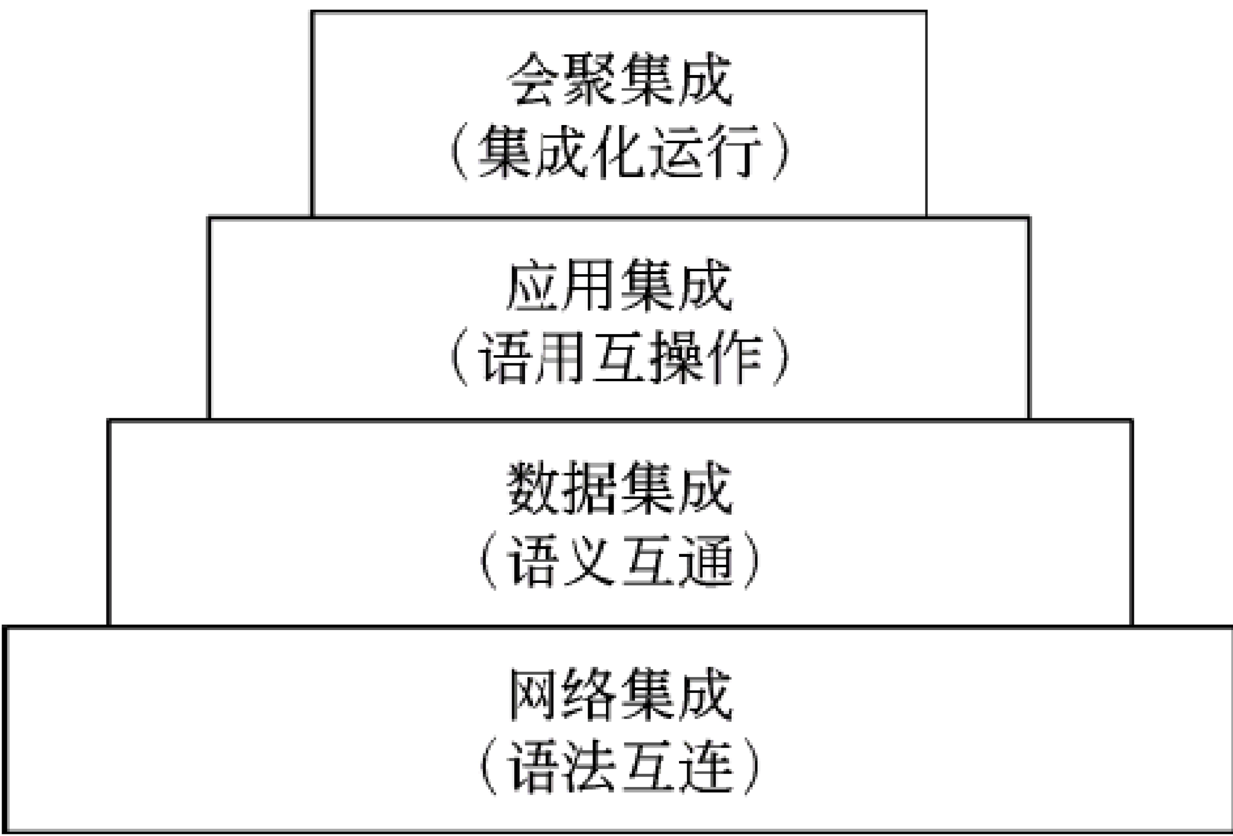


图 17-8　企业集成技术架构层次图

数据集成主要有以下三种模式：数据联邦、数据复制和基于接口的数据集成。如图 17-9 所示，它们分别描述了对多个异构数据源透明、一致访问的三种实现方法。

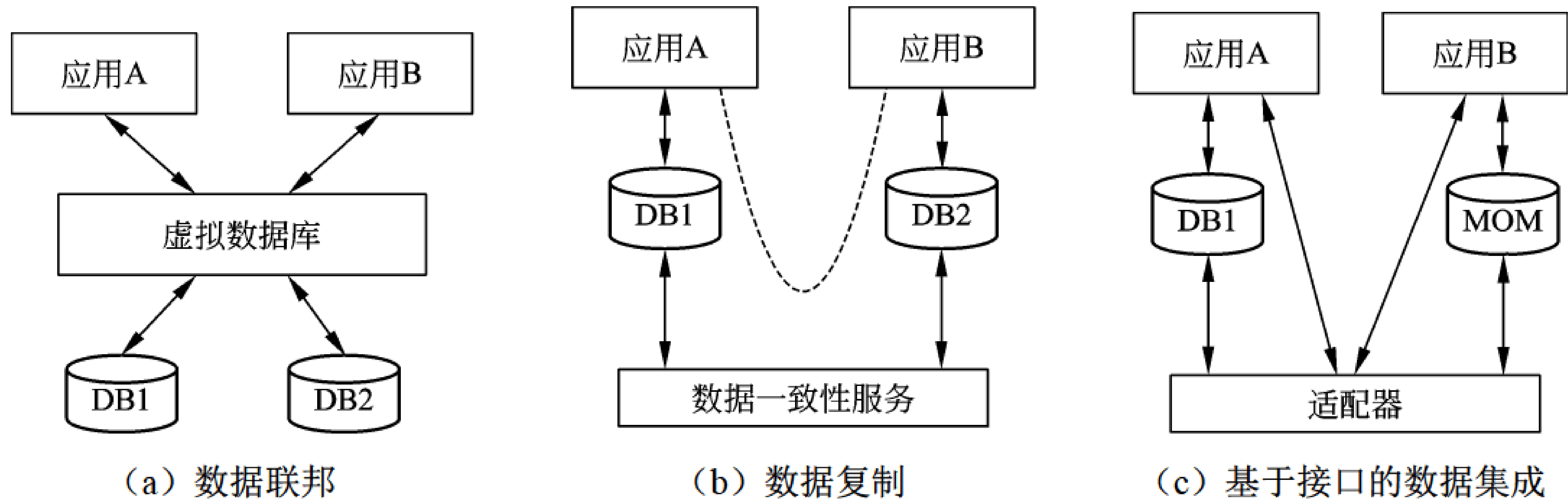


图 17-9　三种典型的数据集成模式

1．数据联邦

数据联邦是指不同的应用共同访问一个全局虚拟数据库，通过全局虚拟数据库管理系统为不同的应用提供全局信息服务，实现不同的应用和数据源之间的信息共享和数据交换，其具体实现由客户端应用、全局信息服务和若干个局部数据源三部分组成。

2．数据复制模式

在数据复制模式中，通过底层应用数据源之间的一致性复制来实现（访问不同数据库的）不同应用之间的信息共享和互操作，其实现的关键是必须能够提供在两个或多个数据库系统之间实现数据转换和传输的基础结构（以屏蔽不同数据库间数据模型的差异）。

3．基于接口的数据集成模式

在基于接口的数据集成模式中，不同的应用系统之间利用适配器（或接口代理）提供的应用编程接口来实现相互调用。应用适配器或接口代理通过其开放或私有接口将业务信息从其所封装的具体应用系统中提取出来，进而实现不同的应用系统之间业务数据的共享与互交换。接口调用的方式可以采用同步调用方法，也可以采用基于消息中间件的异步方法来实现。

17.2.2 应用集成

应用集成是指两个或多个应用系统根据业务逻辑的需要而进行的功能之间的相互调用和互操作。应用集成需要在数据集成的基础上完成。应用集成在底层的网络集成和数据集成的基础上实现异构应用系统之间语用层次上的互操作。它们共同构成了实现企业集成化运行最顶层会聚集成所需要的，技术层次上的基础支持。

应用集成最初主要采用点对点的紧耦合方式。这种集成方式虽然不需要对应用系统做较大的改动，但用这种方式集成的系统缺乏必要的柔性，不能适应业务系统快速重构的需求。随着应用软件系统设计和实现过程中标准化程度的不断提高，系统的开放性（可配置性、可扩展性）越来越好，组件化的系统实现及松散耦合（它是实现系统柔性的基础）的应用集成方式逐渐成为构建企业业务处理系统的主流。

应用集成模式包括集成适配器、集成信使、集成面板和集成代理 4 种，每种应用集成模式都是对具有业务功能依赖关系的多个应用之间互操作实现方法的总结。在具体应用中，集成模式可能以某种变形（这是一种扩展集成模式的主要方式）的形式出现，这些变形可能不仅仅只是一种模式的实例化，也可能是一种具有广泛适用性的集成方式。

1．适配器集成模式

在 EAI 技术发展的初期，广泛采用在需要交互的系统之间加入适配器（Adapter）的解决方案来实现企业原有应用系统与新实施系统之间的互操作。在应用系统提供的 API 的基础上（在应用系统没有提供 API 的情况下，可以在其数据库表结构已知的条件下直

接完成对其数据库的写入与读出)，通过适配器完成不同的系统间数据格式及访问方式的转换与映射，进而实现不同的系统之间业务功能及业务数据的集成，如图 17-10 所示。

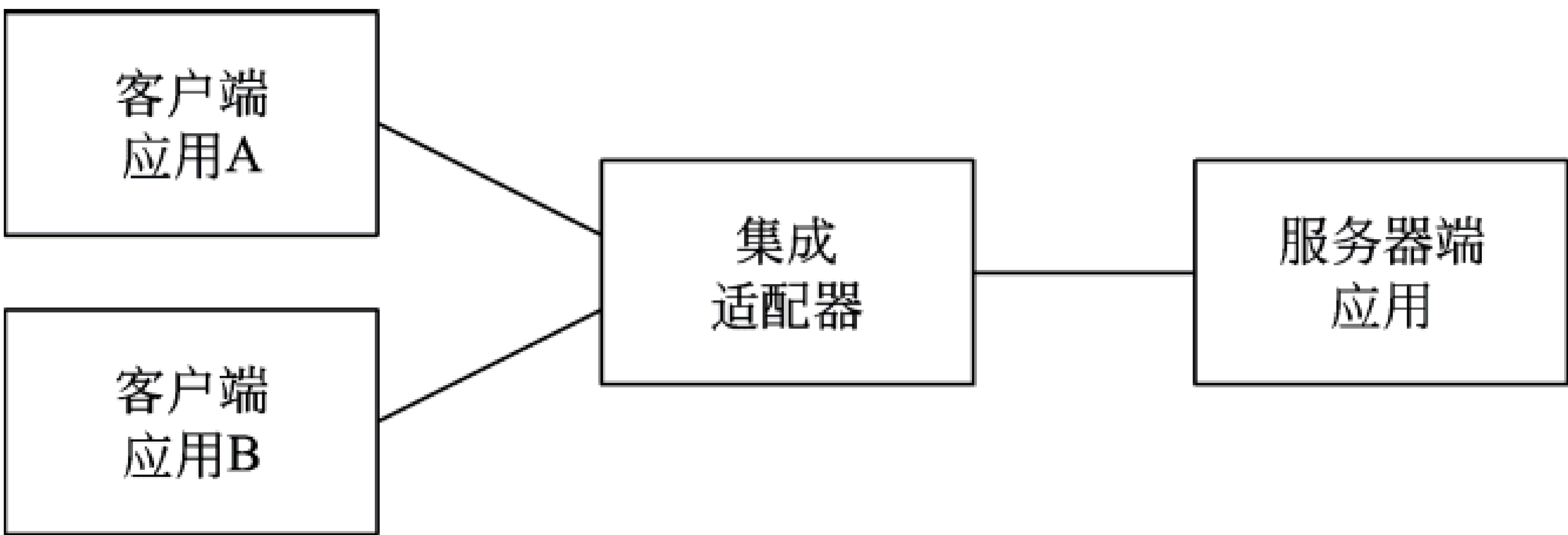


图 17-10　适配器集成模式

2．信使集成模式

随着企业中业务应用系统个数的增多，应用系统间的接口问题变得越来越复杂。为了更灵活地实现应用系统间点对点的集成问题，提出了图 17-11 所示的基于信使的集成结构。在这种集成结构中，系统之间的通信和数据交换通过信使（消息代理）来实现，每个应用只需要建立与集成信使之间的接口连接，就可实现与所有通过集成信使相联的应用系统间的交互。这种结构大大减少了接口连接数量，同时由于采用了信使（消息代理）作为信息交流的中介，可以将应用之间的交互对通信服务能力的依赖程度降到最低。另外，当某一系统发生改变时、只需要改变信使中相应的部分，从而降低系统维护工作量和系统升级的难度。

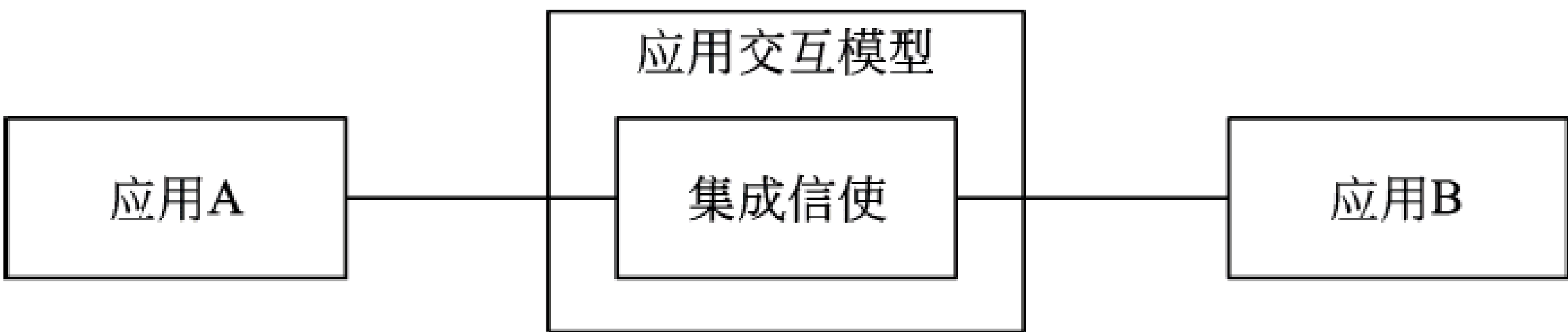


图 17-11　信使集成模式

3．面板集成模式

面板集成模式和面向对象的软件设计方法中的面板模式很相似，它是从应用交互实现的层面来描述客户端应用和服务器端应用集成的一种方法。图 17-12 给出了面板集成模式框架图。集成面板可以为一对多、多对一、多对多等多种应用提供集成接口，在这种模式中包含有一个或多个客户端应用、一个集成面板、一个或多个服务器端应用。集成面板通过对服务器端应用功能的抽象和简化，为客户端应用访问与调用服务器端应用提供了一种简化的公共接口。集成面板在得到客户端应用服务请求后，将客户端的服务请求转换成服务器端应用能理解的形式，并将该请求提交给服务器端应用。

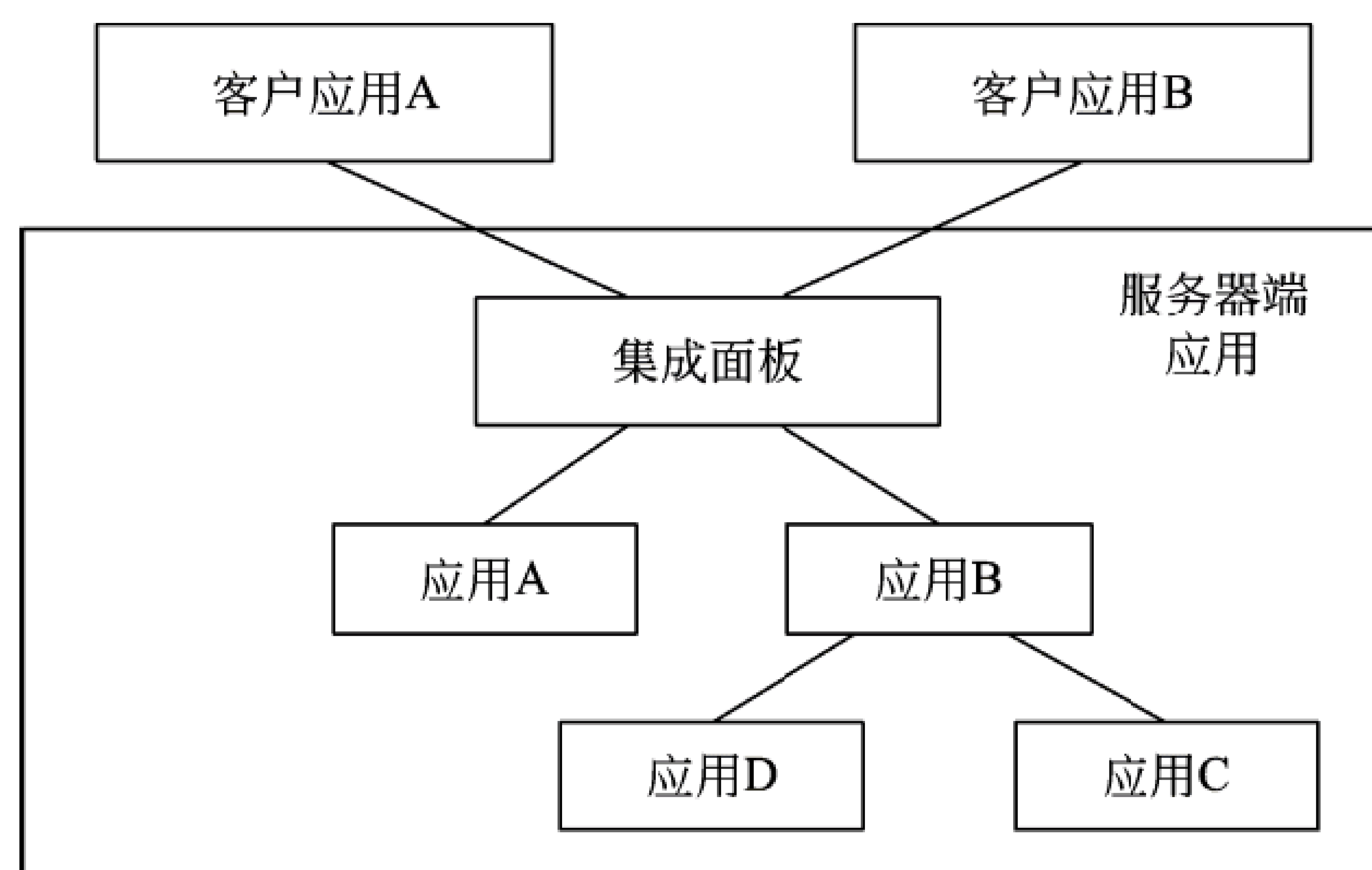


图 17-12　面板集成模式

4．代理集成模式

面板集成模式实现了服务器端应用交互逻辑的分离。在代理集成模式中，由于不存在很明显的客户端应用和服务器端应用的划分，它仅需要将待集成的应用间的交互逻辑从应用中分离出来，并对应用间的交互逻辑进行封装，进而由集成代理来引导多个应用之间的交互，如图 17-13 所示。

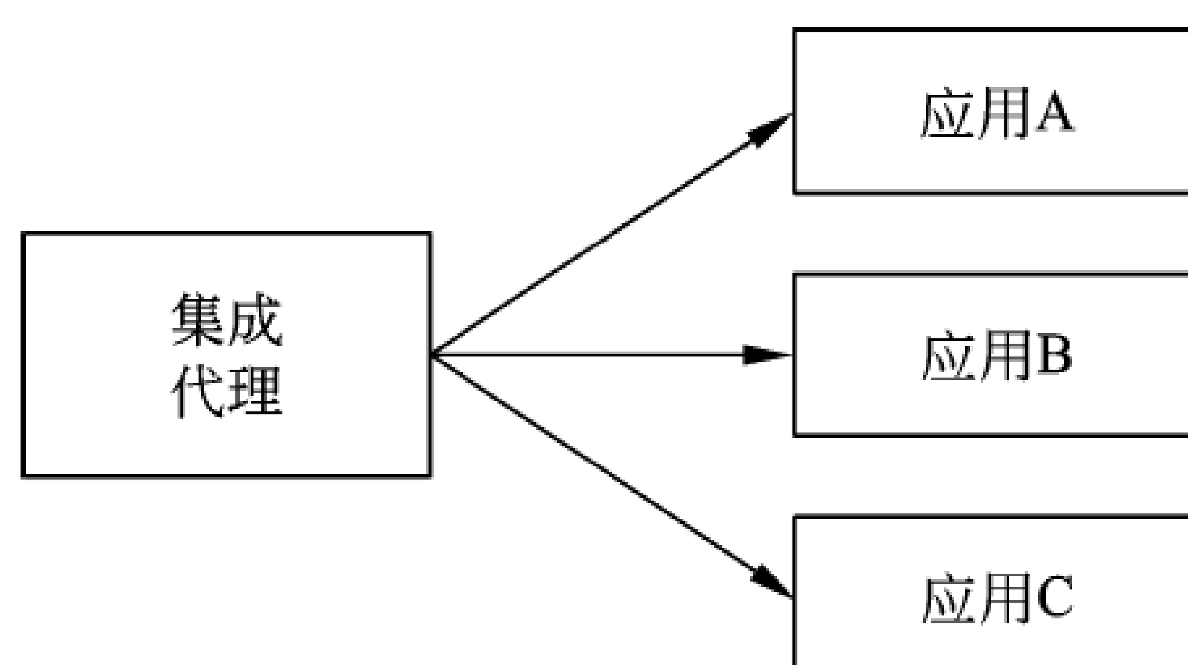


图 17-13　代理集成模式

17.2.3　企业集成

企业应用软件系统从功能逻辑上可以分为表示、业务逻辑和数据三个层次，其中表示层负责完成系统与用户交互的接口（界面）定义，业务逻辑层主要根据具体业务规则完成相应业务数据的处理，数据层负责存储由业务逻辑层处理或产生的业务数据，它是系统中相对稳定的部分。按照这些逻辑功能层次间是否分离和分离的程度，在软件系统具体实现上可以大致分为如下 4 类。

（1）单层结构系统。

很多企业遗留应用系统属于这一类，这种应用一般是采用传统的编程方法得到的一个紧密结构应用，三个层次之间没有进行分离，因此某个层次的变化通常需要重新设计

与开发其他两个层次的内容。

（2）两层结构系统。

通常是将表示层与业务逻辑层（胖客户）紧密地耦合在一起，或者是将业务逻辑和数据库层紧密地耦合在一起（只将表示层分离出来为瘦客户）。这种结构实现了三个层次间部分的分离，这样在应用的某个部分发生变化时仅需要修改与其紧密耦合的部分，而无需重新开发所有的代码。如将表示层分离出来，可以使同样的业务功能采用不同的图形化用户接口及显示器屏幕模式，改变客户端接口（如增加 Web 界面）并不需要修改业务的逻辑功能来实现。

（3）三层结构系统。

这是当前比较流行的系统实现方式。它将业务应用系统的表示、业务逻辑和数据三个层次分成独立的模块实现。这样，应用系统的各层可以并行开发，各层也可以选择各自最适合的开发环境和编程语言。这种系统结构不但提高了系统的可维护性，也有利于系统的安全管理。

（4）n 层结构系统。

将三层系统结构进一步细化（主要是将业务逻辑及数据库层分成更多、粒度更小的分布式业务对象来分别实现），其目的是提高系统不同业务功能模块的独立性。在提高了系统的可配置能力的同时，可以使系统具有最好的柔性及可扩展能力。

支持企业间应用集成和交互的集成平台在系统结构上通常都采用多层的结构，其目的是在最大程度上提高系统的柔性。在集成平台的具体设计开发中，还需要按照功能的通用性程度（通用功能、面向特定领域的功能、专业化功能）对系统实现模块进行分层（分成不同的中间件）。

根据企业集成平台功能的支持范围，可以将其划分为侧重于支持企业内部集成化运行的 EAI 和侧重于支持企业间业务集成的 B2B。一般来说，EAI 是 B2B 的基础，下面主要讨论 EAl 的实现模式。

从企业集成运行的实现策略上看，EAI 主要有如下三种实现模式。

（1）前端集成模式。

所谓前端集成模式，是指 EAI 侧重于业务应用系统表示层的集成，它主要通过单一的用户入口实现跨多个应用事务的运作。这种方式适合于用户启动的业务过程会产生多个跨应用的事务，而且这些事务都需要实时响应的情况（主要指 B2C 的环境）。另外，采用前端集成模式还可以实现对已经运行的核心业务应用系统增加功能或特征的目的。

（2）后端集成模式。

后端集成模式主要侧重于应用系统数据层面的集成。它通过专门的数据维护及转换工具实现不同应用或数据源之间的信息交换，维护企业整体业务数据的完整性和一致性。

后端集成模式就像一个方便多个应用系统之间数据自动交互的数据管道，后端集成模式的实施同样需要得到数据集成及应用集成的支持。后端集成模式实现起来相对比较

简单，因为 EAI 服务器不需要跨应用的事务维护，而只需要维护一些相对简单的业务规则。基于 EAI 服务器提供的存储——转发机制可以方便地实现对合作伙伴企业之间大量业务数据交换（主要指 B2B 集成）的支持。

（3）混合集成模式。

混合集成模式是前端集成模式和后端集成模式的组合。客户通过基于 Web 浏览器的客户端（瘦客户）实现对业务应用或 EAI 服务器的访问，服务请求可以由前端应用系统执行，也可以通过 EAI 服务器将服务请求路由到后端，由后端的业务应用来执行。这种模式几乎具有前端集成模式和后端集成模式的所有特征，主要应用于既需要响应大量服务请求、又需要维护多个数据源的完整性和一致性的情况。

17.3 企业集成的关键应用技术

17.3.1 数据交换格式

企业业务数据可以分为结构化数据（表单）和非结构化数据（文档），它们一般存储在不同的数据库或文档管理系统中。不同的应用系统、数据库所处理的文档和数据格式有很大差别，建立各个应用都可以识别和访问的通用数据模型及表示规范，是实现不同的应用系统之间交互和互操作的最基本方法。企业数据集成中常用的几种数据交换格式如下。

1．EDI

EDI（Electronic Data Interchange，电子数据交换）是一种利用计算机进行商务处理的方法，它将贸易、运输、保险、银行和海关等行业的信息，用一种国际公认的标准格式，通过计算机通信网络，供有关部门、公司与企业之间进行数据交换与处理，并完成以贸易为中心的全部业务过程。

EDI 格式处理的目的是将在功效上与纸介质文件等同的电子表单用统一的（或标准的）格式进行表示，以保证各个独立开发的计算机应用间能够实现表单数据共享与集成。用于描述电子表单格式的标准称为 EDI 格式标准或 EDI 标准，目前广泛使用的 EDI 格式标准主要有 UN/EDIFACT 和 ANSIX12，分别由联合国欧洲经济委员会（The United Nations Economic Commission for Europe，UN/ECE）和美国国家标准化协会（American National Standard Institute，ANSI）制定。

国际标准化组织采用 UN/EDIFACT 作为国际标准（IS09735）。按照 UN/EDIFACT 标准，贸易伙伴之间一次交换的内容称为一个交换，交换由交换头 / 尾、功能组头 / 尾、报文头 / 尾、数据段（或段组）和数据元（简单数据元和复合数据元）等组成。为简化起见，数据段（或段组）、数据元等在本文中都被称为报文项。图 17-14 给出了 EDIFACT 报文的数据结构。

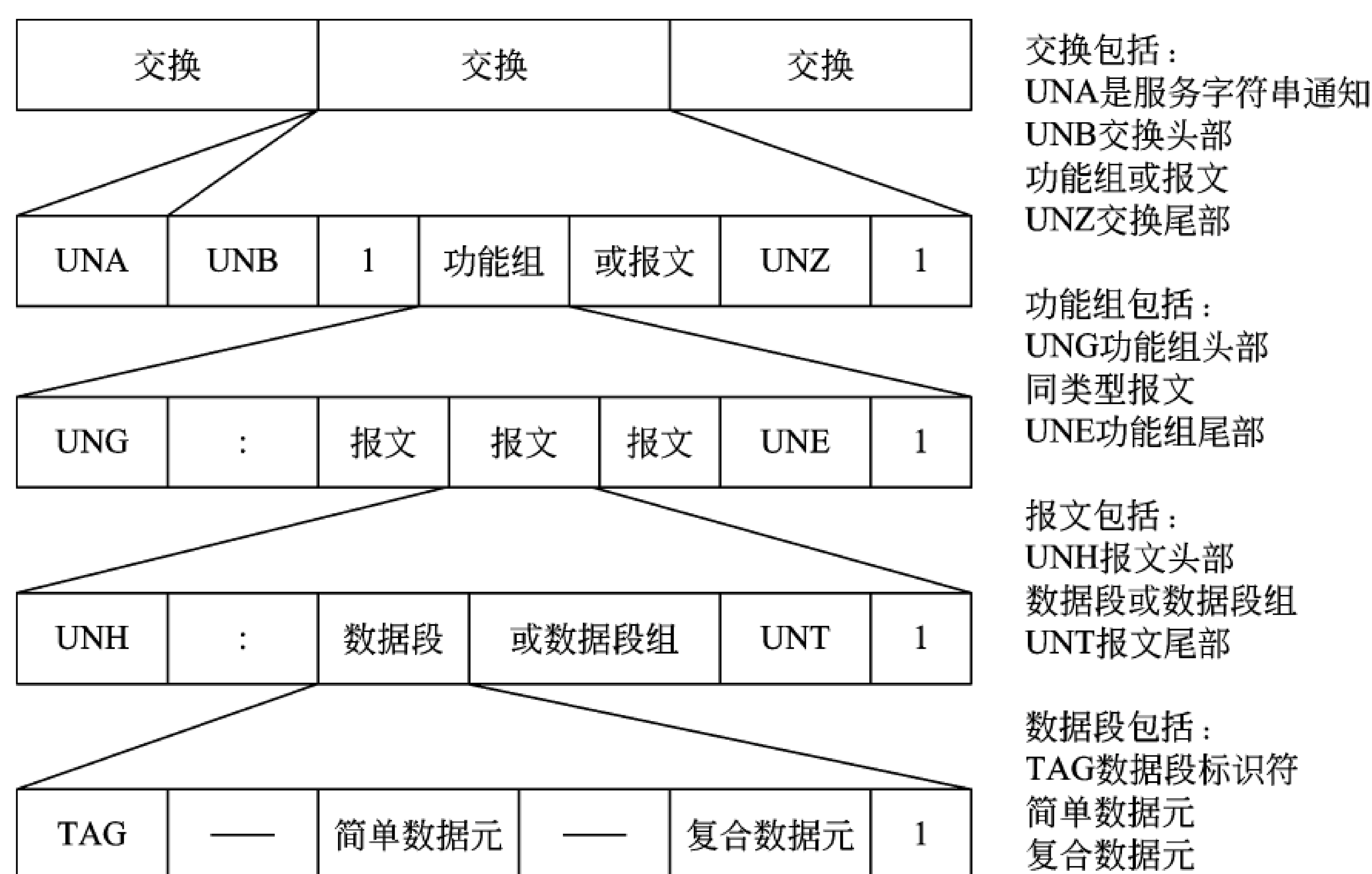


图 17-14　EDIFACT 报文的数据结构

2．XML

XML 是国际组织 W3C 制定的一个面向各类信息的数据存储工具和可配置载体的开放式标准。提出 XML 的目的是为了更好地适应 Web 应用的需求，解决 HTML 在表达能力、可扩展性和交互性等方面的缺陷。XML 是通过对 SGML 标准进行简化而形成的元标记语言，具有语法清晰简单和结构无歧义等优点。它利用一套定义标记的规则将文件的内容和外观进行分离，实现了 XML 文档的可延伸性及自我描述特性，从而使各种业务信息可以在全球信息网或企业间的应用系统中传递、处理及储存。这里需要指出的是，虽然 XML 称为可扩展标记语言，但它本身并不是一种标记语言，而是一种创建、设计和使用标记语言的根规则集，是一种创建标记语言（如 HTML）的元语言。图 17-15 给出了 XML 相关标准的层次图。

3．STEP

STEP 标准（Standard for the Exchange of Product Model Data）是一个描述如何表达和交换数字化产品信息的 ISO 标准（IS010303），其目的是提供一种不依赖于具体系统的中性模型和机制，并将其用来描述整个生命周期内的产品数据。

图 17-16 给出了 STEP 标准的结构，其核心由描述产品数据的形式化语言规范（描述方法）、STEP 实现方法、集成资源和一致性测试标准 4 部分组成，而围绕该核心定义的各种应用协议及抽象测试套件构成了对 STEP 的外层支持。描述方法用于集成资源的定义，由集成资源模型产生应用协议，应用协议和实现方法相结合产生一种 STEP 实现，一致性测试则用于测试 STEP 实现是否与 STEP 标准相一致。

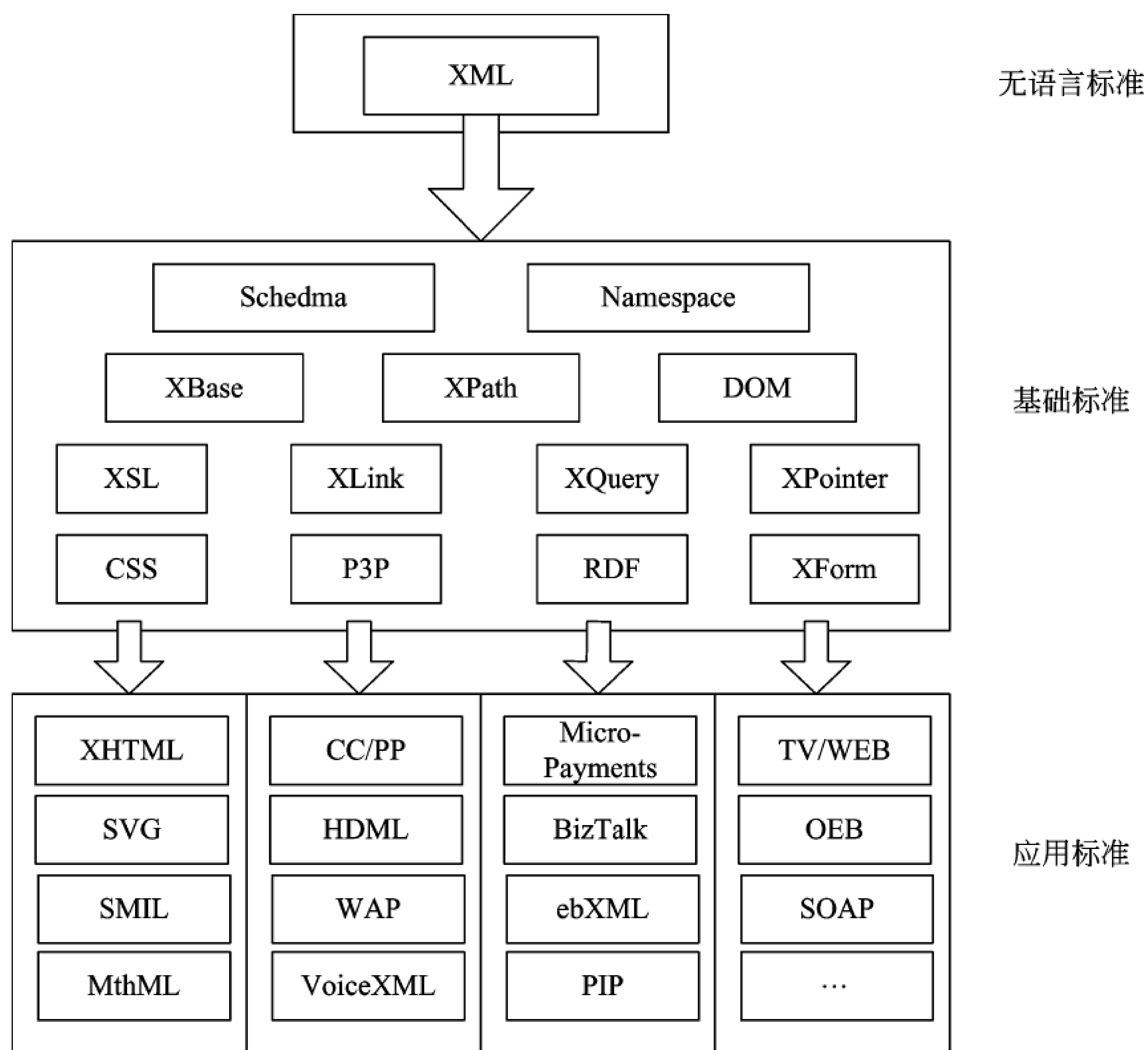


图 17-15 XML 标准体系

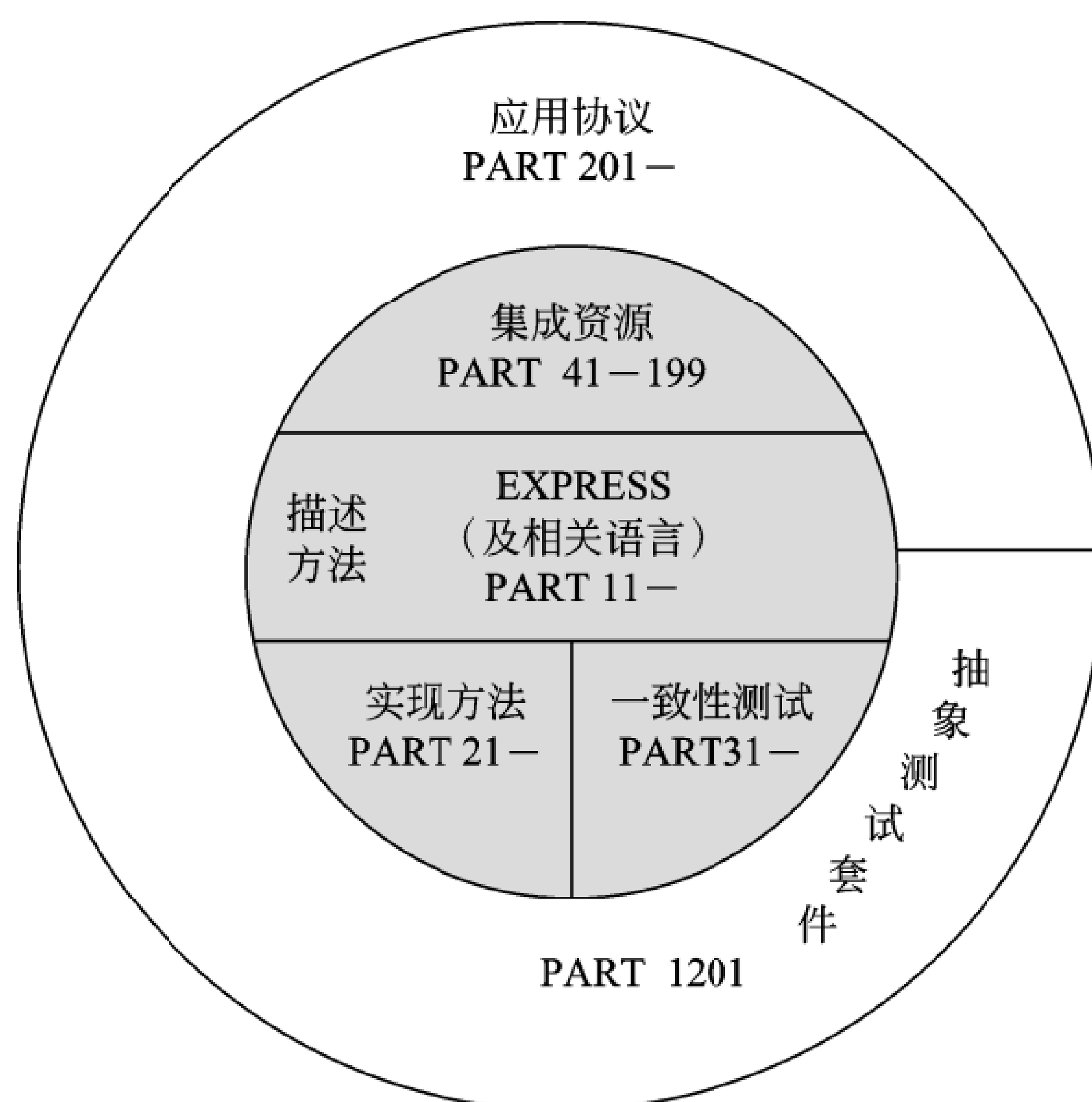


图 17-16 SETP 标准的结构

4．PDML

PDML 的技术目标是提供一种灵活的方法，使得不同应用软件系统中的产品数据能够进行交换。它是在 STEP 和 XML 基础上实现不同系统间产品数据交换和集成的一种新模式。

PDML 中主要应用了 STEP 的集成资源和 EXPRESS 数据规范语言两个部分。在 PDML 中，与特定领域词汇表（或数据字典）相应的组件被称为应用事务集（Application Service Set，ATS），与跨多个应用领域的通用词汇表相应的组件被称为集成方案，集成方案的设计基于 STEP 的集成资源。

PDML 使用 XML 来描述所有业务应用软件系统中的产品数据，并通过提供一系列的标准 DTD 来进行产品数据的导入和导出。由于 EXPRESS 在（产品相关的）语义和约束的表达能力方面要比 XML 的 DTD 优越很多，因此 EXPRESS 被选择作为定义 PDML 模式的规范。为了充分利用 EXPRESS 语言在数据建模和 XML 语言在数据交换方面的优点，PDML 定义了一个从 EXPRESS 模式到 XMLDTD 的转换机制。

PDML 不是单一的产品数据规范，而是一个用来发布和使用集成产品数据的相关标准和工具的集合。PDML 由 7 个应用事务集、一个集成大纲、应用事务集和集成大纲间的映射规范、PDML 工具集 4 部分组成。图 17-17 给出了应用事务集、集成大纲和映射规范之间的关系。

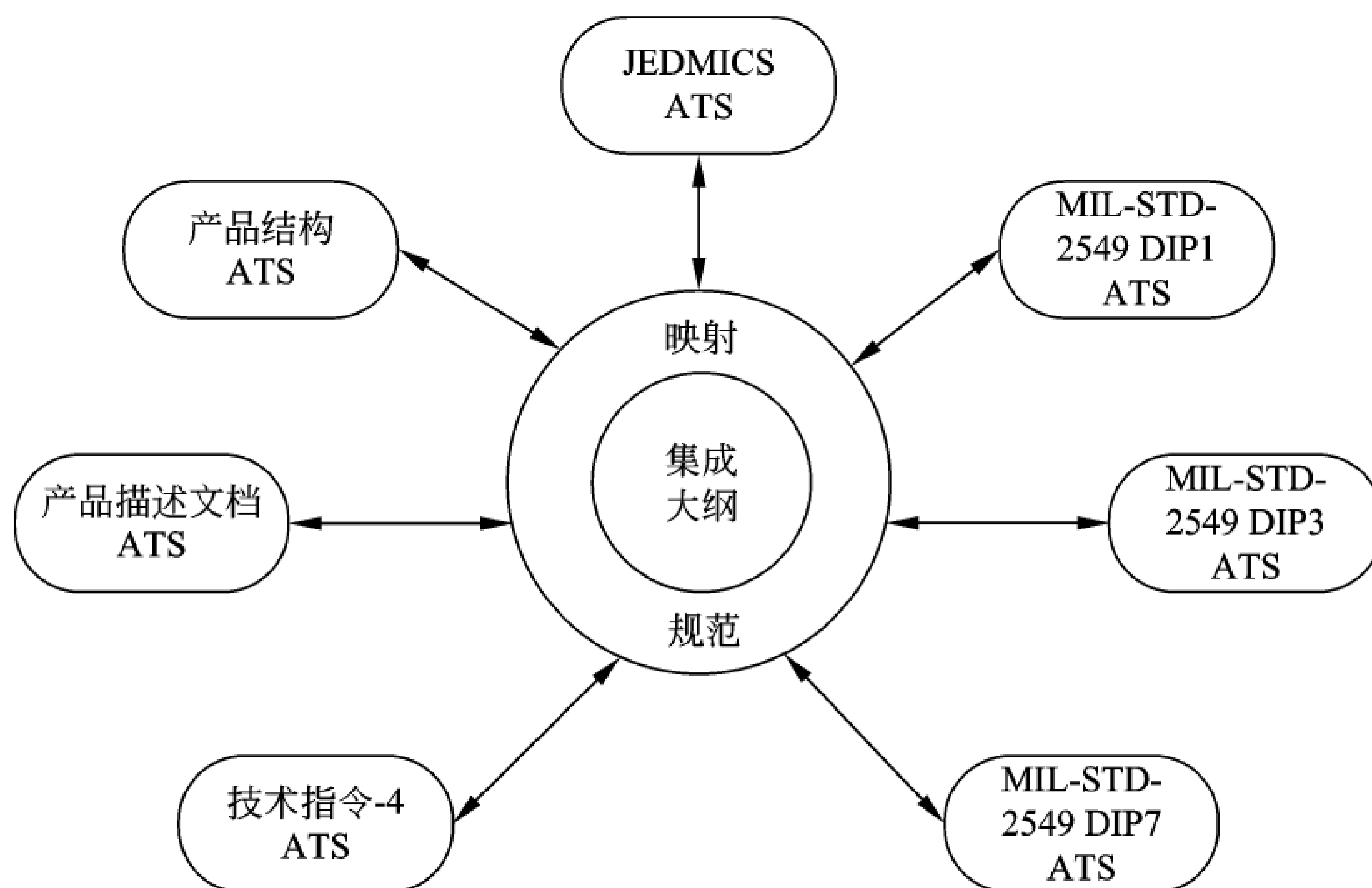


图 17-17　PDML 各组成部分之间的关系

17.3.2　分布式应用集成基础框架

随着计算机网络应用的不断深入和普及，大规模的计算机网络将不断增加，在这种

计算机网络中，不仅硬件设备型号、种类、规模相异，而且操作系统平台、程序设计环境及应用也各不相同，这就是大规模计算机网络的重要特征——异构性。人们迫切希望通过在这种计算机网络上建立一套体系结构和一组规范来保证分布式系统的互操作性、可迁移性和可重用性，进而实现分布式环境下的信息共享与应用集成。因此，在面向对象技术和分布式计算基础上产生的分布式对象计算（Distributed Object Computing，DOC），成为 20 世纪 90 年代计算机技术发展的一个热点。而在当今众多的分布式对象技术中，比较有影响的分布式软件对象（组件）标准有下面三种。

1．CORBA

CORBA（Common Object Request Broker Architecture，公共对象请求代理体系结构）是对象管理组织（OMG）为解决分布式处理环境中硬件和软件系统的互连而提出的一种标准的面向对象应用程序体系规范。

OMG 组织给出了分布计算的参考模型，称为对象管理参考模型（Object Management Architecture，OMA）。OMA 模型中把软件作为对象，并通过对象请求代理与其他对象进行通信。其体系结构如图 17-18 所示。

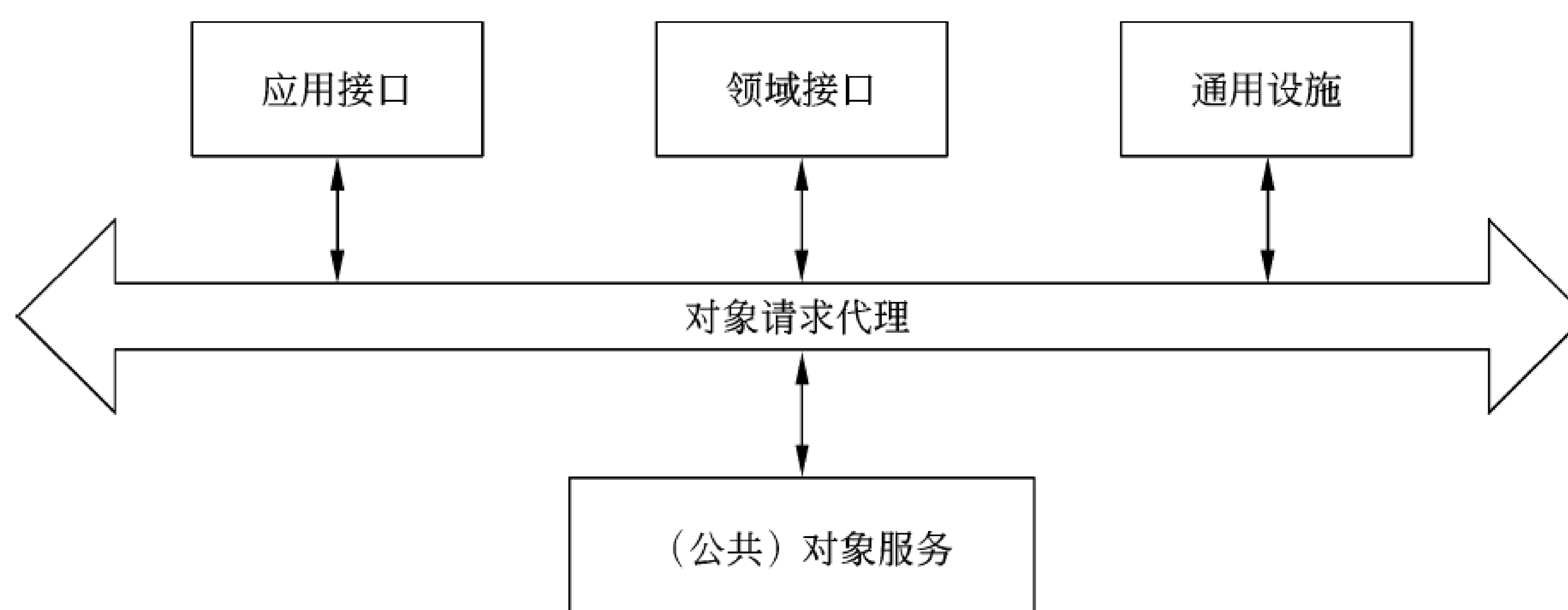


图 17-18　对象管理参考模型的体系结构

OMA 体系结构的核心是对象请求代理（Object Request Broker，ORB），CORBA 规范对 ORB 的组成和功能进行了定义，它支持对象服务、通用设施、领域接口和应用接口之间的交互和通信。

ORB 是 CORBA 的对象互操作中介，作为应用对象间服务请求响应的中间代理，接收对象请求并把请求转给相应的对象，服务完成后又把执行结果或异常情况返回给请求者。ORB 可以使对象以语言、位置和平台独立的方式发出请求和提供服务，相互协同工作，从而建立真正的分布处理，是实现分布对象互操作的核心。COBAR ORB 的组成结构如图 17-19 所示。

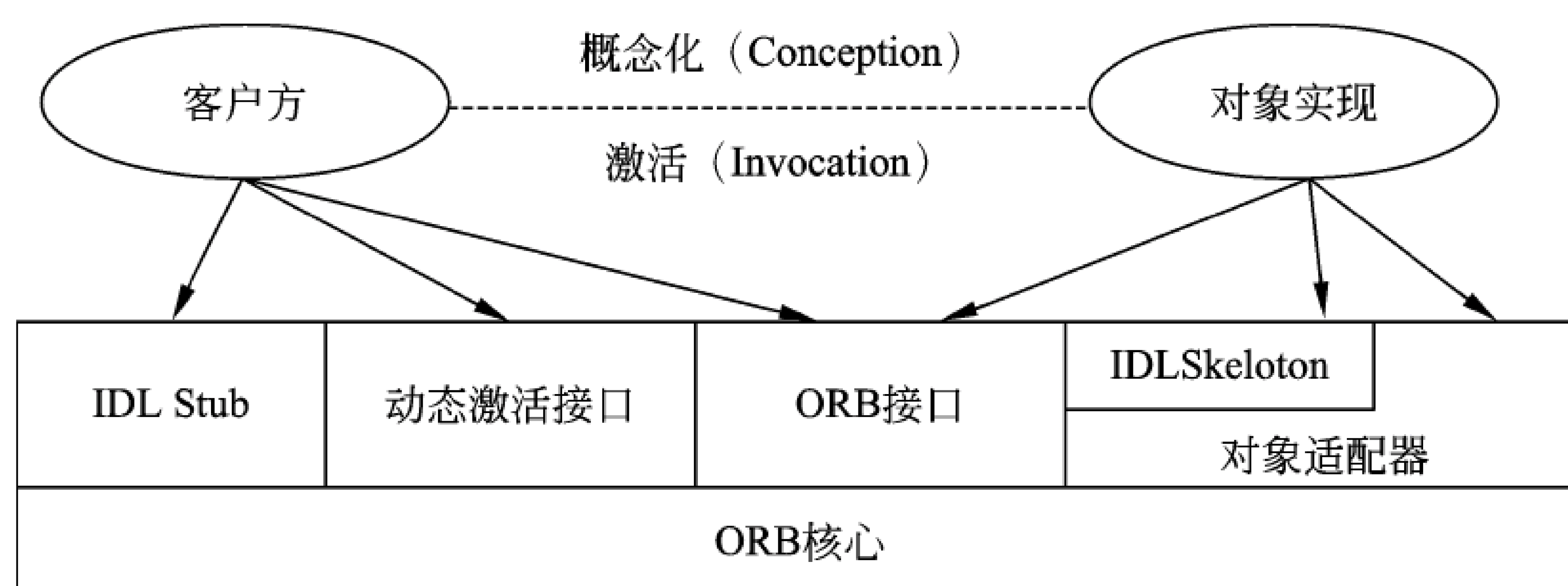


图 17-19　CORBA ORB 结构

2．COM＋

COM +是 Microsoft 公司基于 Windows 平台的一个分布式企业应用模型，它与 Windows 操作系统紧密结合，是沿着 DDE-OLE-OLE2-COM-DOOM-COM＋的路线发展而来。目前，COM、DCOM 和 COM +应用比较广泛。

COM 是一个开放的组件标准，有很强的扩充和扩展能力。COM 组件标准的基础是 COM 核心，它规定了组件对象与客户通过二进制接口标准进行交互的原则。COM 主要由 COM 接口、COM 对象、COM 服务器、类工厂和类型库等组成。其中，COM 接口是和 COM 对象之间互相调用相关的一组语义规范，每个接口有一个唯一标识（UUID）；COM 对象则为一个或多个 COM 接口提供具体的服务（功能实现），对 COM 对象的调用是通过一个指向其接口的指针实现的；COM 服务器提供 COM 运行的环境，完成 COM 对象的管理，并向 COM 客户提供服务；类工厂则是用于创建、注册 COM 对象的特殊对象，它为 COM 对象的实例化提供一种标准机制；类型库是一个二进制资源文件，包含 COM 服务器中对象与接口的类型信息。在 COM 系统中，客户对组件对象功能的调用接口一般采用 COM IDL 来描述。COM 定义了两类服务器，即进程内服务器和进程外服务器。进程内服务器即本地机上的 DLL，进程外服务器分为两类：一是本地机上的 EXE 可执行程序，二是远程机上的 DLL 或 EXE 程序。服务器内部包括组件接口的实现和类工厂，类工厂生产组件对象，将对象的接口指针返回给客户。组件服务器的定位由 COM 库完成并返回对象指针。COM 对象位置的透明性处理由 COM 的服务控制机制保证。进程外的对象必须先调用服务控制机制提供的代理，代理生成服务对象的远程过程调用（Remote Process Call，RPC）。基于 COM 的系统调用原理如图 17-20 所示。

另外，COM 组件标准还包括结构化存储、统一数据传输和智能命名等。其中结构化存储定义了复合文档的存储格式以及创建文档的接口，统一数据传输约定了组件之间数据交换的标准接口，智能命名则给予对象一个系统可识别的唯一标识。COM 组件标准为 COM 对象之间的相互操作奠定了基础。

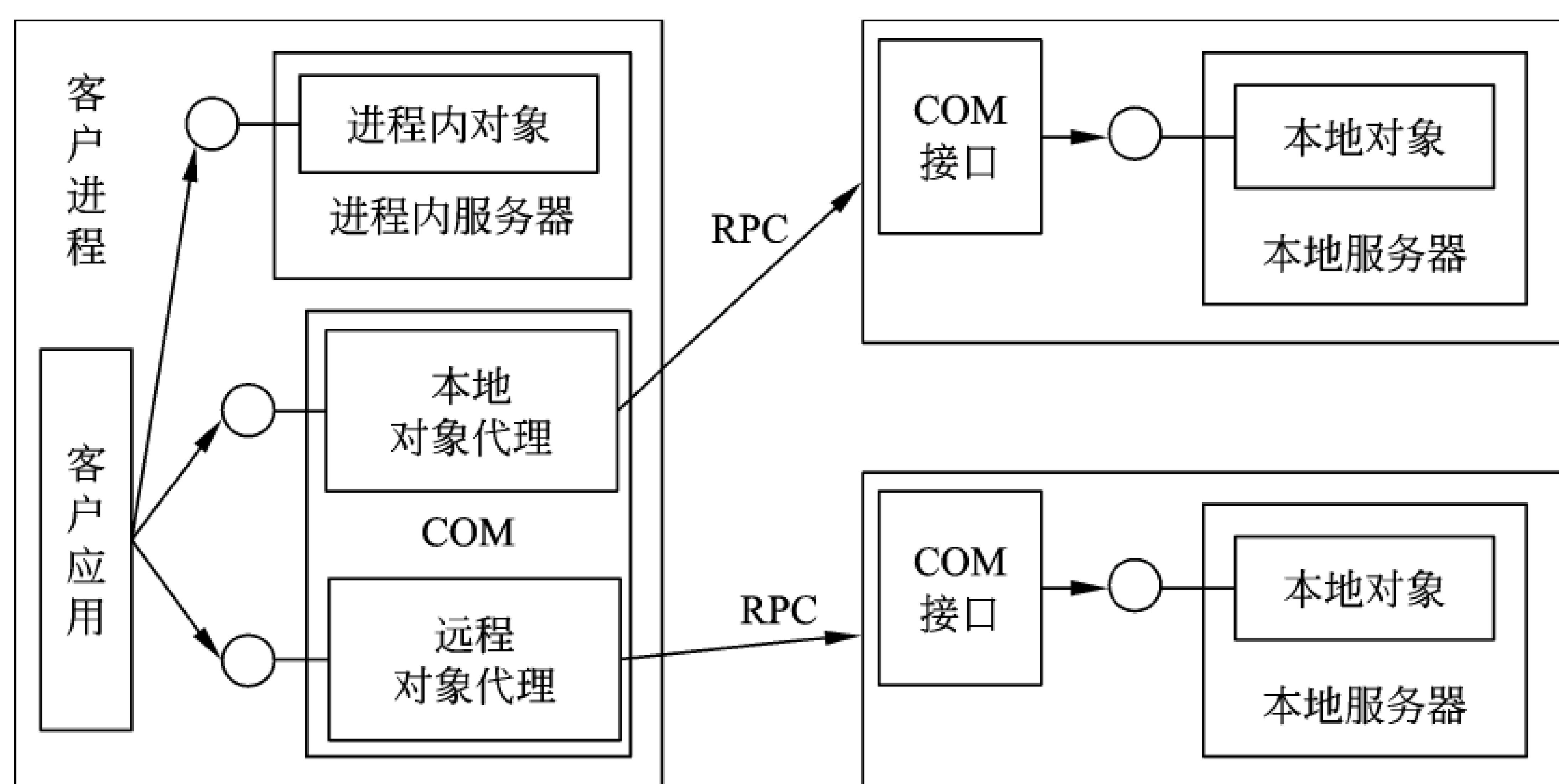


图 17-20 COM 调用原理

3．J2EE

J2EE（Java 2 Platform Enterprise Edition，Java 2 平台企业版）是由 Sun 公司制定的基于 Java 技术的分布式组件计算平台规范。

Sun 设计 J2EE 的初衷是为了解决两层模式的弊端，即系统难于升级或改进、可扩展性差，而且经常基于某种专有的协议。它使得重用业务逻辑和界面逻辑非常困难。J2EE 将两层化系统模型中的不同层面切分成许多层，从而形成了一个多层的端到端的分布式应用系统架构。在图 17-21 给出的基于 J2EE 标准的典型运行结构中，主要包含客户层、Web 层、业务逻辑层和数据层（包含遗留系统）4 个层次。

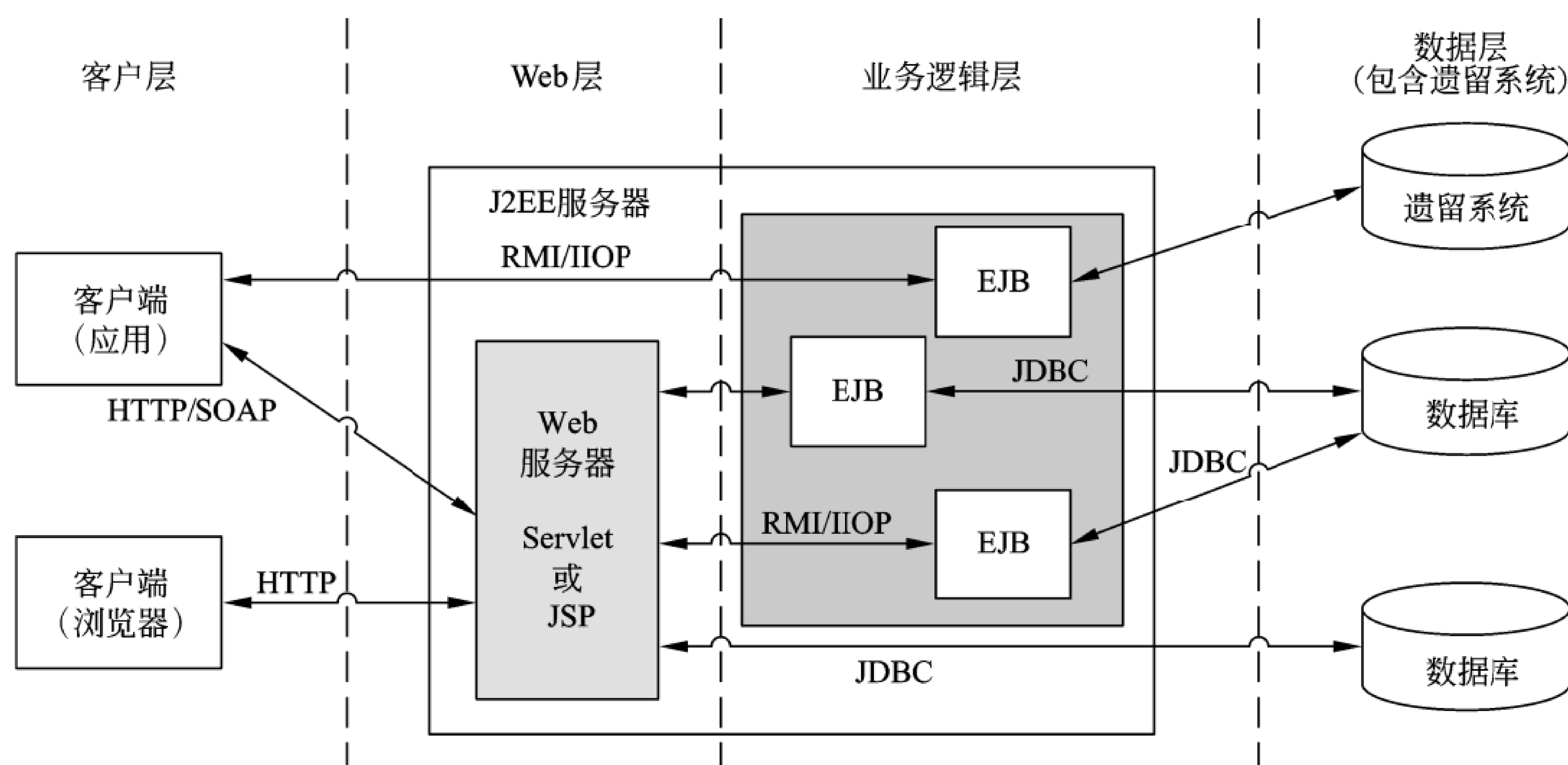


图 17-21 J2EE 运行结构

J2EE 很好地融合了 Internet 技术，有利于企业建立基于 Web、具有 n 层结构的分布式应用，同时它也为应用系统集成提供了良好的解决办法。J2EE 的应用集成架构如图 17-22 所示。J2EE 的基础是核心 Java 平台或 Java2 平台的标准版，J2EE 将 J2SE 集成到自己的体系结构中，不仅巩固了标准版中的许多优点，同时也使 J2EE 供应商能够独立于操作系统与硬件平台来实现应用程序产品。各种组件可以通过 J2EE 配置工具将其部署到相应的 J2EE 容器中，客户端对各种组件的访问及各种组件之间的调用都通过容器及服务器来完成。

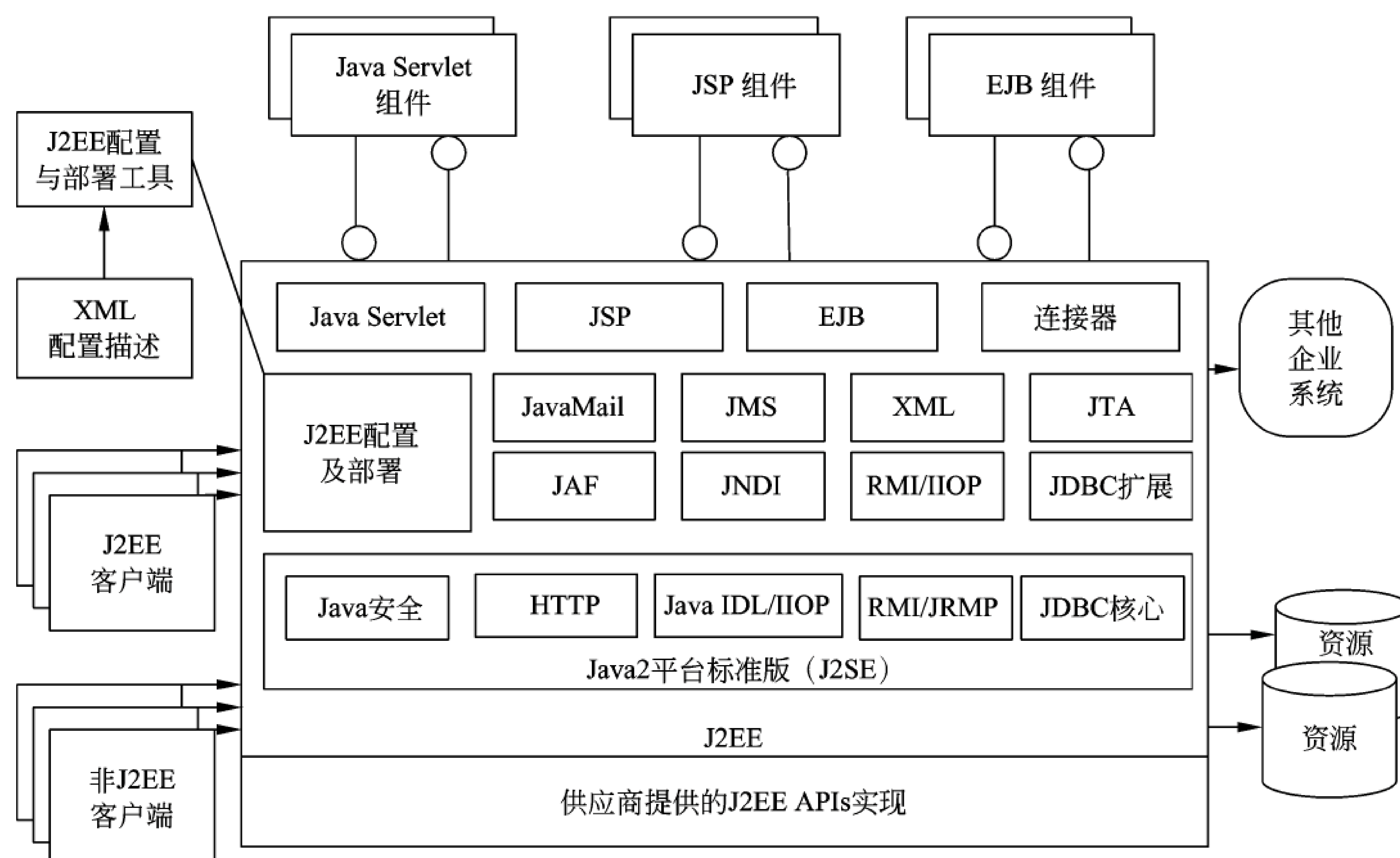


图 17-22　基于 J2EE 的应用集成架构

4．Web Service

Web Service（Web 服务）是指服务提供者将应用作为服务部署在 Web 上，通过使用 Web 服务描述语言来描述特定 Web 服务提供的功能。服务请求者在需要一种 Web 服务时，可以通过 Internet，在 Web 服务的注册机构中查找分布在 Web 站点上的 Web 服务，并自动实现与服务的绑定，完成数据交换，在这个过程中无须人工干预。Web 服务的工作原理如图 17-23 所示。由于 Web 服务的系统架构和实现技术基本上基于已有的技术，因此，Web 服务可以看成是现有应用面向 Internet 的一个延伸。

实现 Web 服务需要相关技术标准的支持，目前支持 Web 服务的技术标准主要有：用于进行数据交换和表达的元语言标准 XML，XML 用来在 Web 服务中表示服务请求和应答的内容；UDDI（Universal Description，Discovery＆Integration），UDDI 用于 Web 服

务注册和服务查找；WSDL，WSDL 用于描述 Web 服务的接口和操作功能；SOAP（Simple Object Access Protocol），SOAP 为建立 Web 服务和服务请求之间的通信提供支持。图 17-24 给出了支持 Web 服务实现的体系结构。

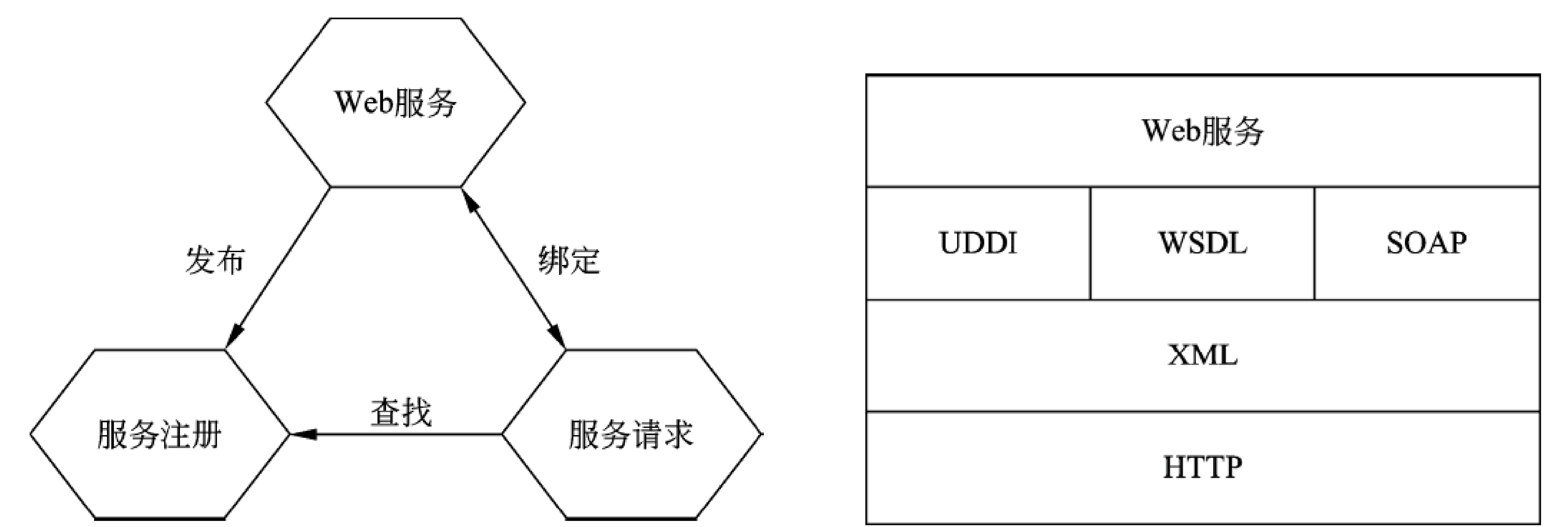


图 17-23　Web 服务的发布、请求和绑定过程　　图 17-24　Web 服务的体系结构

17.4　面向整体解决方案的企业模型

17.4.1　企业模型在整体解决方案中的作用

企业模型是人们了解企业并经过抽象得到的对于企业某个或者某些方面的描述，它是实施企业信息化工程与实现企业集成的基础。企业建模在企业信息化整体解决方案中发挥的作用主要表现在以下几个方面。

（1）企业模型可以为信息化整体解决方案提供对企业公共一致的、规范的表达和描述。

模型为信息化工作中的所有人员提供一个公共的企业表达，所有的规划和决策人员可以站在同一个理解层面上讨论信息化的开展和实施，同时也为信息系统或部件的设计提供一个公共的模型规范，避免在每个信息系统设计时都直接去抽取需要的数据，减少由于这种工作方式带来的不同信息系统反映的企业数据的不一致问题。

（2）建模和基于模型的分析是企业信息化工作的入手点和建立有效的实施途径的基础。

实施企业信息化，首先必须要明确信息化的目的和范围。信息化应该从企业中最迫切需要改革、最影响和制约企业业务目标实现的环节开始，因此企业信息化实施的第一步应该是企业诊断。企业建模是有效并准确地进行企业诊断和分析的必要基础，通过模型来总结概括企业的现状，使信息化工作建立在一个具体、准确的需求的基础上。通过建模过程以及基于模型的诊断来辅助发现企业生产经营中需要解决的企业瓶颈问题和实

现企业战略目标的业务需求，指明信息化需要解决的企业实际问题，为企业决策提供科学的支持。

（3）建模可以对信息系统规划方案进行预评价。

信息化工程是一项风险工程，会牵涉到企业的过程、组织、人员和资源等方面。在企业诊断之后，要进行企业信息化规划，对信息系统方案进行选择和论证。企业建模可以用于建立企业的改进模型，并基于对改进模型的分析来评价改进的效果以及对整个企业的影响。信息化过程也是企业的一种改进过程，企业建模可以描述按照某种规划方案布置了信息系统后的企业业务运行模型（模拟企业未来业务运作的模型），通过对该模型的仿真分析，并与企业现状模型进行比较，评价这个信息系统规划方案的效果以及需要付出的代价。通过对多种不同方案的比较分析，可以选择一种相对优异的信息系统规划方案。

（4）基于模型的工作流执行可以导航和监控各信息系统之间及信息系统与外界的交互。

面向工作流执行的企业模型可以准确地描述贯穿企业所有信息系统的业务过程，以及过程执行中传递的信息，并且可以定义信息系统交互过程中出现的异常情况的处理过程。在信息化工程进入实施阶段后，企业模型可以对集成的信息系统运行的导航和监控起到一定的支持作用。

由以上各方面看出，可以把整体解决方案的求解问题转化为更加具体的，基于企业模型的整体解决方案的求解。这样，在企业信息化整体解决方案的每个部分中都会包含企业模型、企业建模、模型管理、模型操作、模型标准、模型评价、模型转换和参考模型等相应的内容及工具。

17.4.2　整体解决方案中的企业模型重用

通过企业模型重用可以提高企业建模的效率与效果，进而更好地支持企业信息化整体解决方案的实施。不同的企业虽然在生产经营诸多方面都有其特殊性，但是它们都是企业系统的实例，都具有企业最本质的行为和特征，如为了完成企业的目标，都要进行一系列活动（或过程）。可以将构成企业的所有要素（无论是物质实体还是抽象过程）分成三类：一类是最通用的，适用于任何企业；第二类是在一定范围内通用，例如在一个行业内；第三类是某个企业专有的。对应这种分类，集成化企业建模体系框架中定义了三个实现企业模型重用的通用性层次：通用层、部分通用层和专用层。

（1）通用层：提供了整个集成化企业建模体系结构的基本构成成分，既包括不同的建模阶段、不同的建模视图的基本模型构件，也包括与建模活动相关的约束、规则、术语、服务和协议等。该层次的内容具有最强的通用性，能够广泛地适用于各类企业。

（2）部分通用层：在通用模型层的基础上，以生产经营方式类似的企业为背景，通

过对它们典型业务流程和企业行为特征的分析和提炼，形成一组适合于某一行业的部分通用模型（模板），即行业参考模型。每种行业的部分通用模型拥有该行业中大部分企业共有的典型结构参考模型，它可以适用于这一个行业的所有或大部分企业。

（3）专用层：根据企业实际情况和需求，选择一定的参考模型并进行适当改动，形成适合于一个特定企业的专有模型，该模型仅能够用于所描述的企业。

通用性层次的划分使企业建模活动能够从简单到复杂、从抽象到具体、从一般到特殊逐步进行，形成一个层次化过程。利用模型构件可以组成参考模型，参考模型又可以派生出具体的专用模型，对专用模型再进行抽象后又可以形成新的参考模型。

企业通用模型构件及参考模型是在大量工程应用案例的基础上，对诸多企业的共同特征进行抽取而得到的。模型构件及参考模型库的建立和维护可以为企业信息化工程不同阶段的工作提供有实际应用价值的模型框架基础，并有助于进行企业诊断和模型优化，为提高企业建模质量、缩短企业建模周期、减少企业建模成本提供直接的支持。

企业模型可以采用从零开始的方法来建立，但是这种方法存在建模周期长和建模质量低等问题。因此，基于参考模型建立企业具体的专用模型是较好的方法。其实现过程包括两个阶段：参考模型的选择和参考模型的实例化。其中参考模型的选择具体包括以下几个步骤。

（1）确定企业建模的目标和基本需求。

（2）划定企业建模的范围。企业建模可以覆盖整个企业，也可以覆盖企业的某一部分。

（3）提出候选参考模型。参考模型的选择要依据企业规模相关性、行业相关性、产品相关性、生产经营模式相关性和领域相关性等准则。

（4）确定最终使用的参考模型。在候选参考模型中，经过进一步的分析和评价，最终确定一个或一组参考模型。

参考模型的实例化是在参考模型的基础上完成的，实例化过程在具体操作中可以采用的方法如下。

（1）继承：将参考模型中的模型构件或组件直接继承为企业应用模型的一部分。

（2）剪裁：对选取的参考模型，根据企业建模的目的和范围，进行适当的剪裁，作为企业应用模型的一部分。

（3）细化：在参考模型的基础上，根据企业建模的目的和需求，对模型中的某些部分作进一步的分解、细化和完善。

（4）扩充：按照参考模型的结构，对参考模型没有覆盖的企业建模范围加以扩充，形成企业应用模型。

（5）修改：对参考模型中的某些部分按照企业的实际需要进行修改，或者对参考模型中某些组件进行重组。

在具体的建模过程中，通用层、部分通用层和专用层三个层次又具有相互迭代的关

系。当为某个行业里多个具体企业建立起企业模型后，通过抽取模型中共有的行业特性，可以总结出一个适合于这个行业的参考模型。当行业参考模型的内容非常丰富时，可以从中抽取出一些通用的建模构件。反过来，当拥有了足够多且好用的建模构件后，可以通过这些建模构件来搭建新的参考模型。当拥有了足够完善的参考模型后，可以通过实例化参考模型来快速建立起一个具体的企业模型。图 17-25 给出了这一迭代过程的图示化表示。

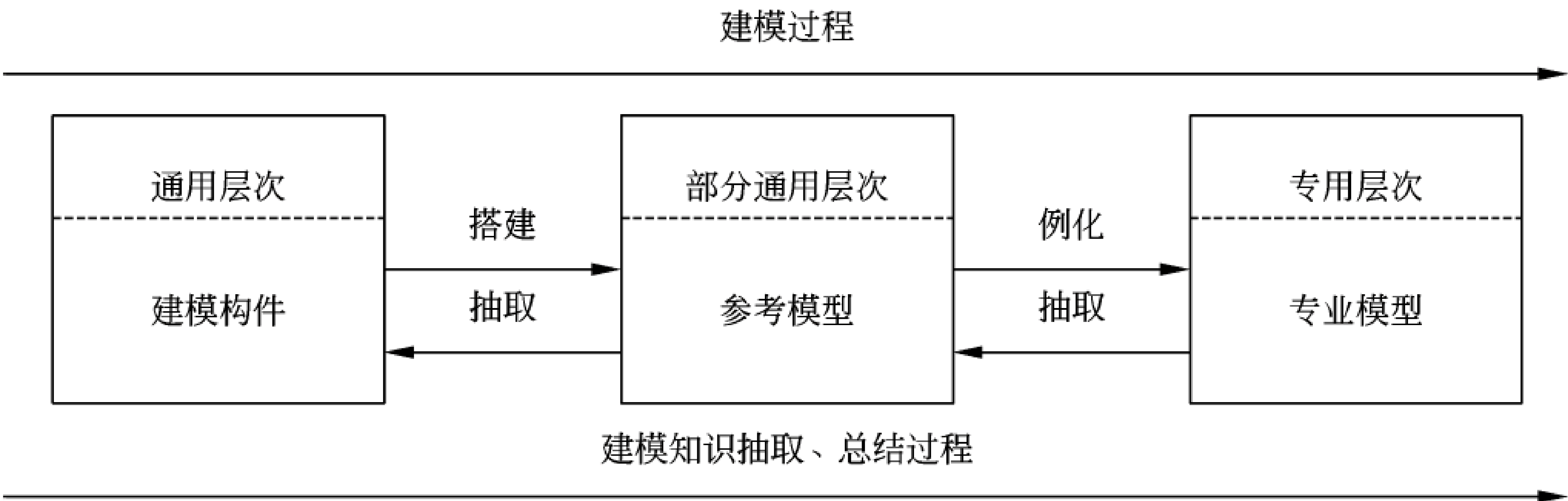


图 17-25　企业模型中的三个层次间的关系

17.4.3　整体解决方案中企业模型演化

企业信息化整体解决方案实施的不同阶段在一定程度上也反映了企业模型及建模过程的阶段性，在不同的阶段，对模型的广度、深度和粒度要求都是不同的，各阶段需要采用哪些视图、各视图采用什么样的描述方法也都会有所不同。所以，在企业信息化整体解决方案的实施中，企业模型处于不断演化的状态之中。信息系统实施的生命周期可以分成需求分析阶段、系统设计阶段、系统实施阶段和运行维护阶段。下面分别介绍这 4 个阶段对企业模型的要求和在建模过程中需要完成的工作。

1．需求分析阶段

需求分析阶段主要完成企业业务策略、信息技术/系统策略的确定与分析，并在完成业务调查及建立企业现状模型的基础上，结合用户需求，发现企业现有的优缺点，并针对缺点和瓶颈提出优化需求以及优化目标。在这一阶段通过对用户需求的抽象形成需求分析模型，以作为下一个阶段的输入。所建立的需求分析模型应该包含有较高层次上的企业业务流程、资源分配、组织结构和产品结构等信息。最后还需要确定系统的总体目标和评价标准。

2．系统设计阶段

在确定了信息系统的需求之后，系统设计阶段则主要完成企业目标模型的确定和信息系统集成框架的求解，从未来的信息系统相关的业务模型中抽取出功能模型和信息模型，用它们来设计和构造信息系统。功能模型描述系统功能的划分和逐级分解，每一个

功能单元对应信息系统的一个功能模块，功能模型是对业务过程模型中过程和活动所实现功能的归纳。信息模型描述信息系统需要使用到的数据结构和数据之间的关系，为建立信息系统数据库进行概念建模和物理建模。信息模型中的内容也来源于需求分析阶段建立的业务核心模型。

3．系统实施阶段

系统实施阶段主要完成整体解决方案指导下的信息系统构建，将企业集成框架物化为实现企业集成化运作的协同信息系统。这一阶段实现了企业模型从设计模型向可执行模型的转化。在设计模型的基础上，通过定义具体的操作者、执行器、资源实体、组织单元和应用软件等，形成系统的实施模型。在给定的软硬件和网络环境下，将所得到的实施模型按照系统规划的实施步骤逐步投入运行。具体的工作包括将经过优化后得到的过程模型进行实例化，为业务流程中需要使用的人员、资源和产品指派实际的对象，建立企业信息的物理数据库供实际业务系统使用。

4．运行维护阶段

运行维护阶段则主要完成对投入运行的企业集成化系统的运行维护，通过文档管理、版本控制等方法实现对运行系统的有效管理和监控，并通过集成需求管理软件工具来对运行过程中企业不断提出的新的需求进行记录和管理，所积累的需求和文档是下一个生命周期的输入。

企业的优化是一个持续的过程，一个系统实施后在运行维护阶段搜集的问题和需求又会启动一个新的生命周期。所以整个企业模型演化构成一个闭环，每个阶段的结果（输出）是下一个阶段的输入，上一个生命周期的运行维护阶段得到的结果（输出）是下一个生命周期需求分析阶段的输入。这个不断循环的生命周期以螺旋式上升的形式实现企业相关状态及行为的改进与扩展。企业模型演化的生命周期如图 17-26 所示。

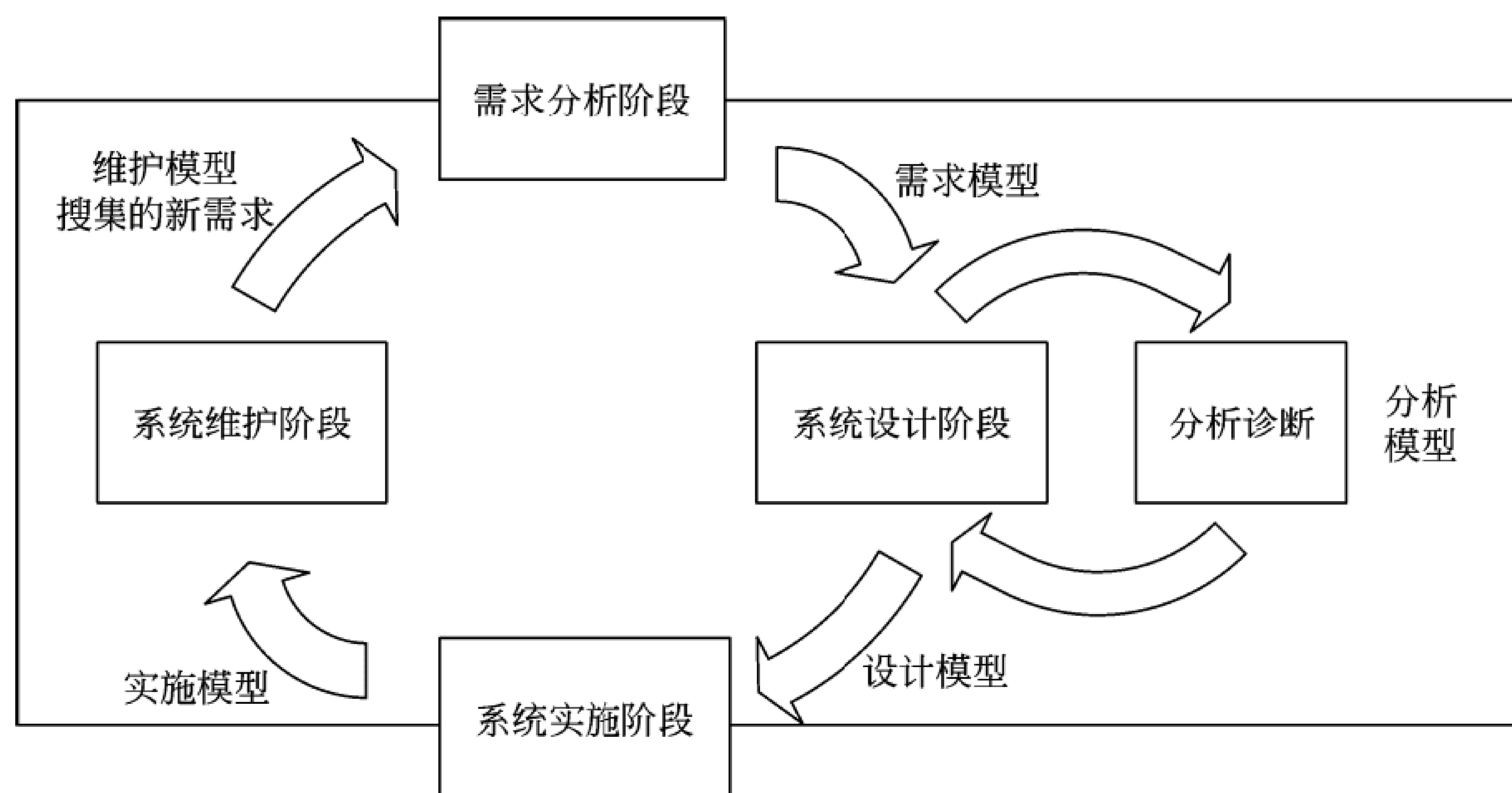


图 17-26　企业模型演化的生命周期

17.4.4　模型驱动的企业集成系统演化

采用企业信息化整体解决方案的目标是通过系统化的理论与方法来指导企业信息系统的规划与实施，构建一个既能够满足当前企业需求、又具有可持续发展能力的集成化业务计算环境。企业可持续发展必然要求支持企业各种资源（包括数据、应用、业务流程、服务及人员等）协同运作的企业集成系统具有可逐步发展和演化的特性。

图 17-27 给出了基于模型的企业集成系统演化模式。首先，经过集成平台实施形成了（根据企业业务模型确定的）企业信息系统集成框架，以及在集成平台支持下的满足企业当前需求的协同信息系统（可能只实现了集成框架下的部分功能）。由于这种实施是根据企业当前的市场策略、业务过程规划和当前的信息技术现状进行的，它只能够在当前的企业和市场状态下，通过信息技术支持企业实现其竞争优势。在这样的集成平台支持下的业务运作，是和企业的业务逻辑（反映市场环境）与业务功能实现技术（反映技术现状）密切相关的。随着市场的变化、技术的进步，企业的核心能力及竞争策略可能要做相应的调整，而这种根据市场、技术的变化调整业务流程或资源配置结构的需求必

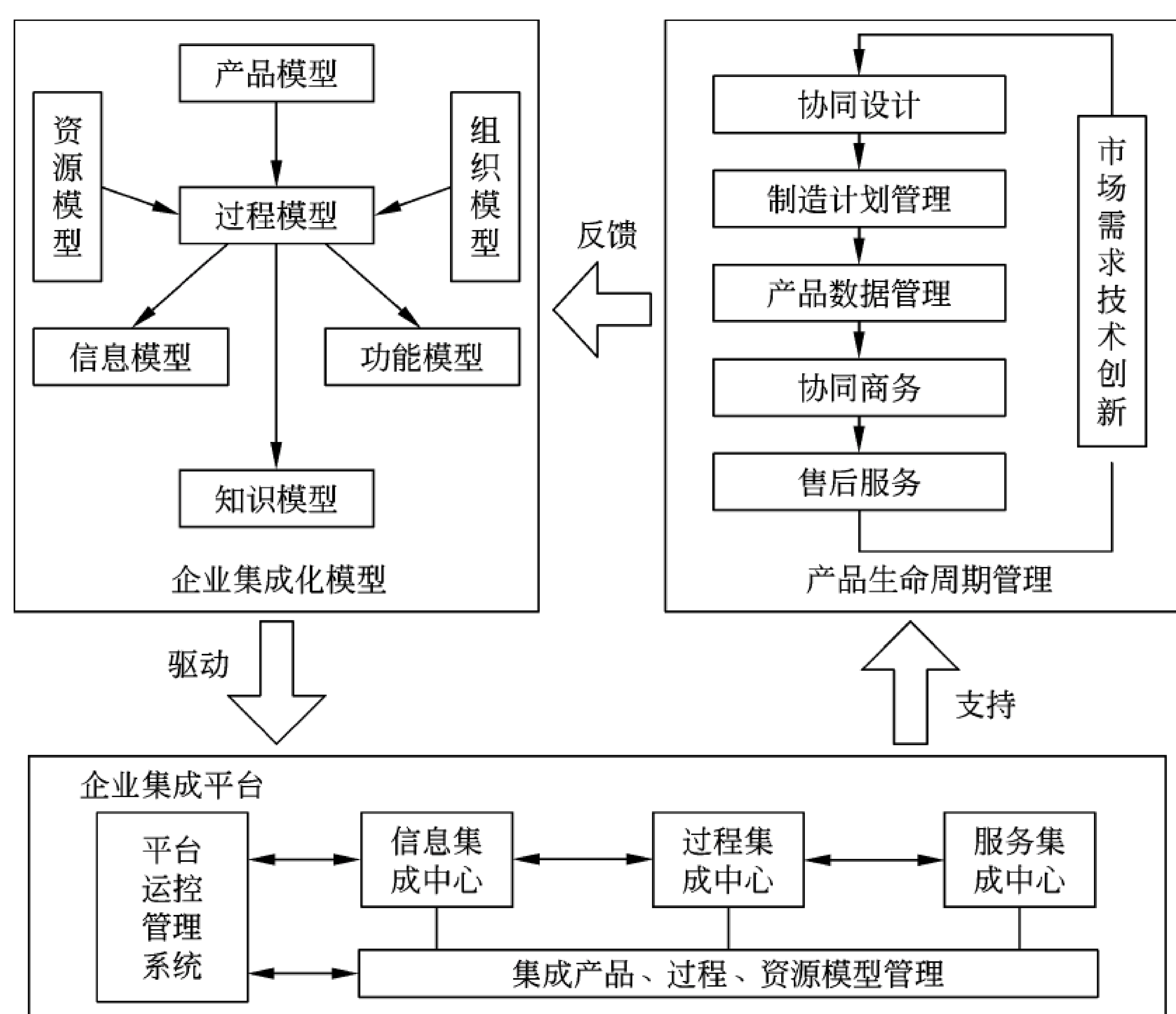


图 17-27　基于模型的企业集成系统演化模式

然要在（在各种信息系统支持下的）业务协同运作过程中得以体现。随着这种需求的不断增加，企业的管理层需要对企业竞争战略或业务流程作必要的调整或改进，从而将这种调整业务模型的需求反馈到（转变为）描述企业业务特征的企业集成化模型中，进而驱动了企业目标模型的演化，企业目标模型的演化又推动了基于模型的企业集成框架和支持业务协同运作的企业集成平台的演化（业务逻辑模型修改或升级信息系统），这个不断循环的过程导致企业信息化工程以螺旋的方式不断上升。

第 18 章　面向方面的编程

随着计算机越来越广泛地应用于社会各个行业，应用软件的规模不断扩大，复杂度不断提高。传统的软件开发方法，如过程化程序设计、面向对象程序设计等已渐渐不能适应这种变化。近年来，一种新的程序开发方法——AOP（Aspect Oriented Programming，面向方面编程）引起了国内外的广泛关注，并被《MIT 技术评论》杂志评为 21 世纪 10 种对经济和人类生活工作方式最具影响力的技术之一。

18.1　方面编程的概念

18.1.1　AOP 产生的背景

1．面向过程编程面临的问题

面向过程编程是一种自顶向下的编程方法，其实质是对软件进行功能性分解。它适用于小型软件系统，例如某一算法的实现。在大型应用系统中，自顶向下逐步求精的方法无论在系统体系结构的确立，系统的进化和维护，以及软件重用性方面都存在其不足之处。

2．传统面向对象编程面临的问题

传统的面向对象语言由于其良好的封装性、层次化性以及继承性等特性而取得了很大的成功，并且对象模型可以很好地映射到实际领域。但是，在软件的生命周期中，它存在以下不足之处。

（1）设计阶段，由于以类为单位组织建模，因此它不能全面地反映软件系统的需求。

（2）编码阶段，将数据和方法封装到类中的思想增强了数据的安全性和软件的模块化，但是有一些数据和方法是特定于应用的，因此这种编码阶段的封装减少了代码重用的可能性。

（3）维护阶段，由于类中夹杂了各种特定于应用的代码，使得维护人员难以理解代码。此外，完成某个特定需求的代码分散在各个类中，当这些代码需要改变时，很难把它们全部找到，这就给程序的健壮性带来了隐患。

由于上述这些问题的产生，需要一种新的程序设计方法从更高的层次上对软件系统进行抽象，将传统的按功能或按对象划分程序模块的方法转化为按系统特征划分程序模块，这就是 AOP 的基本思想。

3．AOP 的产生

在 1997 年的欧洲面向对象编程大会（ECOOP97）上，施乐公司 Palo Alto 研究中心首席科学家、大不列颠哥伦比亚大学教授 Gregor Kiczales 等人首次提出了 AOP 的概念，此后每年的 ECOOP 上都有 AOP 相关的专题研讨会，各大公司、大学、研究机构纷纷投入人员进行研究。2001 年 3 月 15 日，Palo Alto 研究中心发布了首种支持 AOP 的语言 AspectJ。

18.1.2 面向方面的原因

为了理解和完成一个复杂的程序，通常要把程序划分为若干较小的子程序。理想的划分准则已成为众多研究的题目——这些研究的目标对开发人员在程序的设计、发展、维护和更新方面有所帮助。

当一个程序按实现过程编写时，应用程序依照实现的行为和步骤模块化。当使用面向对象的方法时，程序的模块化组则基于类中封装的数据。两种情况下，某些操作较难实现模块化。我们称涉及到这些操作的代码是分散的。

1．代码分散现象

无论是使用面向对象程序设计还是其他方法，代码分散的问题与特定的程序设计语言没有关系，且其影响已经在大量的应用程序中表现出来。事实上，代码分散可能出现在任何编程环境——从 J2SE 或 J2EE 下的 Java，到.NET 下的 C#，到其他语言。但对此现象最广泛的研究是用 Java 实现的。

例如，AspectJ 小组分析了 Tomcat 服务器的容器。他们认识到，如果像 URL 模式匹配和 XML 分析这些操作在一个或两个类中完全模块化，其他操作会高度分散在引用程序中，例如日志功能和对使用者通信的管理。

2．关于代码分散的分析

知道了代码分散的出现，那么是否可以不同地组织类的结构或用其他方法设计程序来消除这个问题呢？

代码分散现象发生的主要原因与服务的可用方式和其使用方式的不同有关。一个类通过它的方法提供一个或多个服务。在同一个类中，聚集可用的服务是相对容易的。然而，一旦这些服务被若干个类所使用，将对这些方法的调用聚集在一起并重新构建这个应用程序会变得困难。因此，一个基本的服务在应用程序中到处被调用就没有什么奇怪的了。

代码分散现象在所有复杂程序中都会表现出来。然而，它的出现实际上依赖于一个具体的问题，代码分散问题很难去除。

应用程序中的代码分散减慢了程序的发展、维护和更新的速度。当若干个操作被分散，情况就会变得更复杂，因为代码包含了许多对多种关系的调用，这些关系逻辑上联系松散但需要结合在一起。

3．一个模块化的新因素

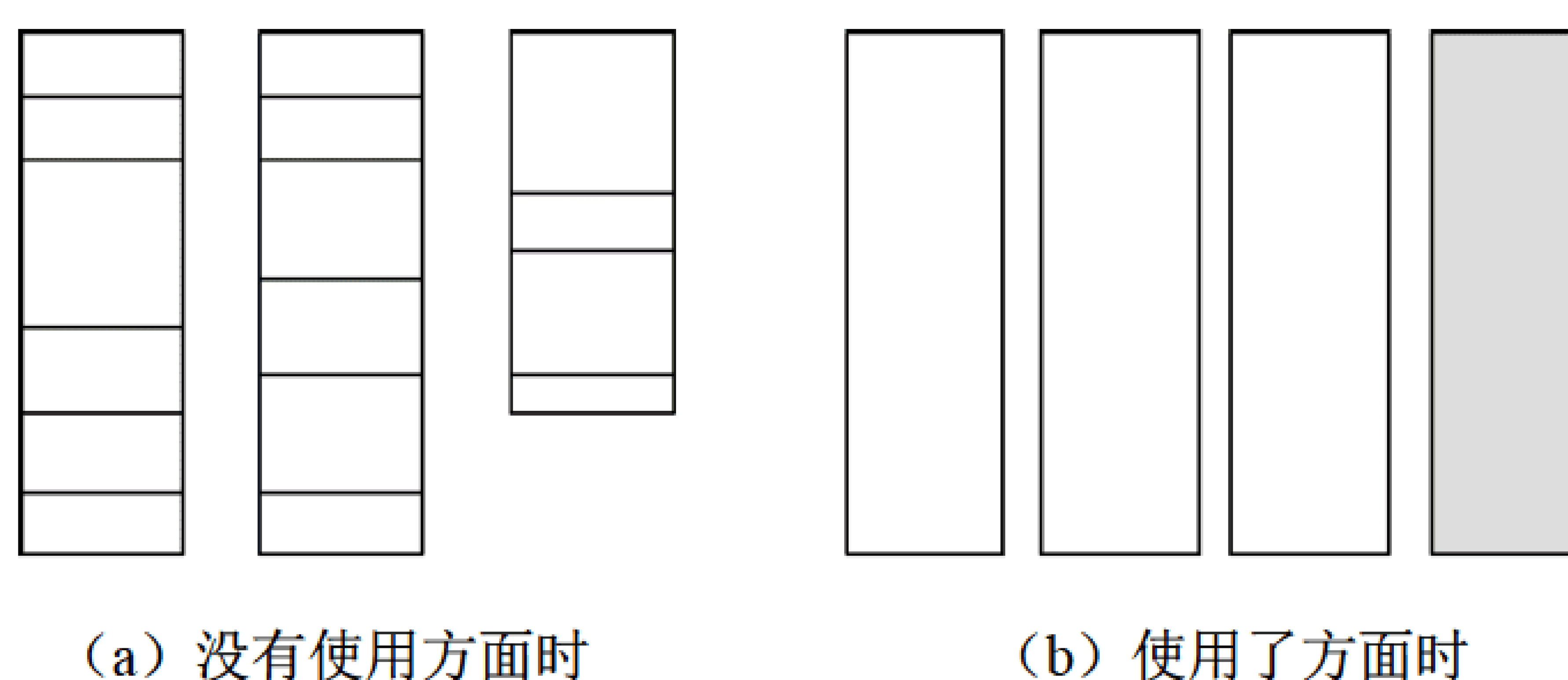
AOP 主要的贡献在于在某一方面提供了一种融合代码的方式——否则这些代码会分散在整个应用程序中。

方面的定义：一个设计来用于捕捉应用程序横切面功能的程序单位。

一个方面通常描述为一个横切程序的结构。实际上，方面这个概念的发明者 Gregor Kiczales 提到，“AOP 是用来捕捉一个横切的结构。”

方面的定义几乎和类一样普通。当对一个问题建模时，人们用类来表示对象（顾客、命令和供应者等）的种类，且每个对象包含适当的数据（属性）和过程（操作）。同样地，方面用于实现一个应用程序中的功能性（安全性、持续性、日志记录等等），而这些功能性要求同样的数据和处理。使用 AOP 时，一个应用程序包含各个类和方面。方面与类的不同在于它实现了横切程序的功能。在面向过程和面向对象的案例中，横切的功能就是那些遍及应用程序的代码。程序中包括类和方面意味着模块性可以在两个因素上实现：类实现基本的功能性（这个因素叫做结构性），方面实现横切的功能性（这个因素叫做可操作性）。

图 18-1 说明了方面在应用程序的代码优化上的作用。图 18-1（a）表现了一个含有三个类的程序。水平线表示代码行相应的横切的功能性，如日志功能。这种功能横切整个应用程序，因其可影响所有类。图 18-1（b）显示使用了方面处理日志功能的同样的程序（带阴影的矩形）。实现这个功能的代码已完全被这个方面所包含，而类则与这些代码分离了。用这种方法设计的程序比没用使用方面的程序容易编写、维护和改编。



（a）没有使用方面时　　（b）使用了方面时

图 18-1　横切功能中方面的影响

4．横切功能的综合

方面由两个部分组成：切入点和通知代码。

通知代码包括要执行的代码，切入点定义了程序中要执行的代码处的点。

显然，方面所包含的代码（或更准确地说，通知代码）依赖于你所要执行的操作。例如，若你想保证数据的持久性，需要在数据库编写保存数据的代码。虽然可以根据基本原理编写这些代码，但你极少会这样做。通常认为的良好的习惯做法是使用一个专门

的 API，例如 Hibernate，通过这种类型的框架，这个方面的代码只是调用了 API。这种工作方式意味着方面并不需要知道服务是怎样执行的，因而方面就与一个特定的执行独立了。

根据这种最优方法，一个方面只允许你整合一个贯穿程序的功能到程序中，这个功能利用一个专门的 API 执行。在图 18-2 中，方面 PersistenceAspect 使用 Hibernate 整合维持数据持续性的功能到类 1 和类 3 中。

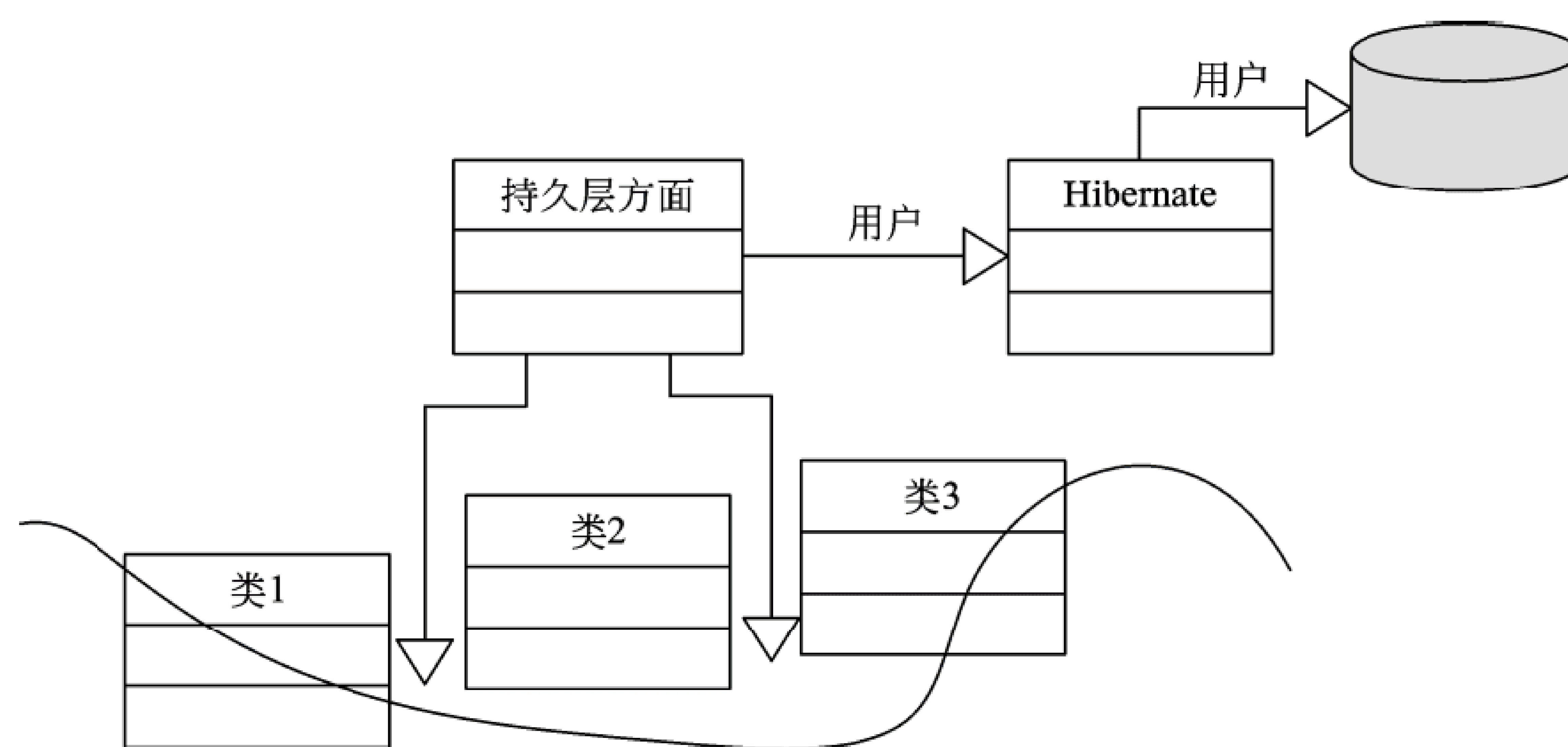


图 18-2 使用方面实现横切的数据持续性功能的综合

严格地说，一个方面并不直接执行一个横切程序的功能，而是使用了一个专门的 API 去实现。但为了使这方面的知识容易理解，仍可以说一个方面执行了一个横切程序的功能。

5. 非功能的服务和方面

大多数应用程序有两种考虑：商业的和非功能的。商业的考虑，也叫做功能上的需求，符合真实世界需要建模的行为。非功能的考虑，或非功能的需求，是附加的服务，这些服务是应用程序必须执行的——事实上，这是出于技术上的或系统级上的考虑。例如，在一个管理人力资源的应用程序中，添加和删除雇员的功能是出于商业上的考虑，而程序安全性和权限的问题是非功能的。

无论如何，在利用这种差别的时候要仔细，因为一个服务可以在一个程序中是非功能性的，但在另一个程序中却是功能性的。很多情况下，非功能性的服务会被遍及各处的商业层面上的代码调用。因此，非功能性的服务在 AOP 中会像方面一样实现，而商业的考虑则会向类一样实现。然而某些情况下，商业的考虑也可能横切程序——使其适合像方面一样实现。

6. 依赖性的颠倒

在面向对象或程序化编程中，一旦程序从 API 使用一个技术服务，设备与服务之间的一种依赖性就建立了，每个程序对 API 外在的调用会发生一种联系。当 API 改变了或

它的语义发展了，整个程序中对它服务的调用就必须作出改变。这种修改有可能是非常昂贵的——尤其是当 API 被用在程序中众多不同的地方时。

另外，要使用 API 还需要理解它的主要原理。要知道应调用什么方法，应按什么顺序调用，应传递哪些参数等。非功能的服务要被包含进每个开发它的新程序中。所以，即使 API 只开发一次，它可能要包含进许多不同的应用程序中。

通过使用 AOP，程序的开发者并不需要担心非功能的服务。方面开发者除了编写提供服务的代码外，还要管理程序中服务的融合。方面开发者的优势在于，专门的方面开发者比一般的程序开发者对服务有着更好的了解，而一般的程序开发者只是 API 的使用者。特别地，方面开发者能确定通过实现服务使用方式的某些约束，使服务的融合是合适的。

18.1.3　AOP 技术

AOP 可以说是 OOP（Object-Oriented Programming，面向对象编程）的补充和完善。OOP 引入封装、继承和多态性等概念来建立一种对象层次结构，用以模拟公共行为的一个集合。当需要为分散的对象引入公共行为时，OOP 则显得无能为力。也就是说，OOP 允许定义从上到下的关系，但并不适合定义从左到右的关系。例如日志功能，日志代码往往水平地散布在所有对象层次中，而与它所散布到的对象的核心功能毫无关系。对于其他类型的代码，如安全性、异常处理和透明的持续性也是如此。这种散布在各处的无关的代码被称为横切（cross-cutting）代码，在 OOP 设计中，它导致了大量代码的重复，而不利于各个模块的重用。

而 AOP 技术则恰恰相反，它利用一种称为“横切”的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为 Aspect，即方面。所谓“方面”，简单地说，就是将那些与业务无关，却被业务模块所共同调用的逻辑或责任封装起来，以减少系统的重复代码，降低模块间的耦合度，并有利于未来的可操作性和可维护性。AOP 代表的是一个横向的关系，如果说“对象”是一个空心的圆柱体，其中封装的是对象的属性和行为，那么面向方面编程的方法，就仿佛一把利刃，将这些空心圆柱体剖开，以获得其内部的消息。而剖开的切面，也就是所谓的“方面”了。然后它又以巧夺天工的妙手将这些剖开的切面复原，不留痕迹。

使用“横切”技术，AOP 把软件系统分为两个部分：核心关注点和横切关注点。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，它们经常发生在核心关注点的多处，而各处都基本相似，例如权限认证、日志、事务处理。AOP 的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。正如 Avanade 公司的高级方案构架师 Adam Magee 所说，AOP 的核心思想就是“将应用程序中的商业逻辑同对其提供支持的通用服务进行分离。”

实现 AOP 的技术，主要分为两大类：一是采用动态代理技术，利用截取消息的方

式，对该消息进行装饰，以取代原有对象行为的执行；二是采用静态织入的方式，引入特定的语法创建“方面”，从而使得编译器可以在编译期间织入有关“方面”的代码。然而殊途同归，实现 AOP 的技术特性却是相同的。

（1）join point（连接点）：是程序执行中的一个精确执行点，例如类中的一个方法。它是一个抽象的概念，在实现 AOP 时，并不需要去定义一个 join point。

（2）point cut（切入点）：本质上是一个捕获连接点的结构。在 AOP 中，可以定义一个 point cut，来捕获相关方法的调用。

（3）advice（通知）：是 point cut 的执行代码，是执行“方面”的具体逻辑。

（4）aspect（方面）：point cut 和 advice 结合起来就是 aspect，它类似于 OOP 中定义的一个类，但它代表的更多的是对象间横向的关系。

（5）introduce（引入）：为对象引入附加的方法或属性，从而达到修改对象结构的目的。有的 AOP 工具又将其称为 mixing。

上述的技术特性组成了基本的 AOP 技术，大多数 AOP 工具均实现了这些技术。它们也可以是研究 AOP 技术的基本术语。

18.1.4 AOP 特性

衡量软件质量高低的要素主要包括可靠性、可扩展性、可重用性、兼容性以及易用性、易维护性等。AOP 作为一种程序设计方法学，关注于提高软件的抽象程度和模块性，从而在很大程度上改善了软件的可扩展性、重用性、易理解性和易维护性，并由此提高影响软件质量的其他因素。下面通过对 OOP 和 AOP 在提高软件可扩展性、可重用性和易理解性、易维护性等方面的能力比较来阐述 AOP 特性。

（1）可扩展性：指软件系统在需求更改时程序的易更改能力。OOP 主要通过提供继承和重载机制来提高软件的可扩展性，因此它的扩展性体现在类一级。AOP 提供系统的扩展机制，通过扩展 Aspect （AspectJ 支持 Aspect 的继承机制）或增加 Aspect，系统相关的各个部分都随之产生变化。由此带来的另一好处是在软件测试中，通过屏蔽某些 Aspect，可以大大简化软件的测试复杂度，提高测试精度。

（2）可重用性：是指某个应用系统中的元素被应用到其他系统的能力。OOP 的类机制作为一种抽象数据类型，提供了比过程化更好的重用性。泛化机制也使可重用性得到很大提高。OOP 所提供的重用性对非特定于系统的功能模块有很好的支持，如对于堆栈的操作或窗口机制的实现等。但在特定于系统的功能模块中，一个类通常包含很多应用系统相关的数据及对其的操作，此时类的重用性变得十分困难。此外，OOP 的重用性也限于类一级，对于不能封装成类的元素，如异常处理等，很难实现有效的重用。AOP 中的系统模块包括系统组件和影响这些组件的特性，通过将实现基本功能的组件和特定于应用的系统特性分离，使得组件（包括类或者函数）的重用性得到提高，并使不能封装为类或函数的系统元素（Aspect）的重用成为可能。

（3）易理解性和易维护性：是影响软件质量的内在因素，它对软件开发人员和维护人员产生影响。在 OOP 中，类机制的引入使其具有比过程化编程更好的模块性，因此也更易于被程序员理解和维护。但是如上所述的代码缠结问题的存在，使 OOP 技术在易理解性和易维护性方面都难有更大的提高。Kiczales 经过统计发现："如果一个他人写的程序有 37 处需要改动，对于一个最优秀的软件开发人员，也大概只能找到 35 个"。而对于 AOP，对一个 aspect 的修改可以通过联结器影响到系统相关的各个部分，从而大大提高了系统的易维护性。另外，对系统特征的模块化封装无疑也能提高程序的易理解性。

18.1.5　AOP 程序设计

1．AOP 程序结构

基于 AOP 的应用程序结构与传统高级语言的应用程序结构基本类似。传统的高级语言系统实现由以下三部分组成。

（1）一种编程语言。

（2）特定于这种语言的编译器。

（3）利用这种语言编写的应用程序。

基于 AOP 的系统实现也有以上三个主要部分，但由于 AOP 中有了动态 aspect 的概念，因此可进一步细化为如下部分。

（1）一种组件语言，一种或多种 aspect 语言。

（2）一个用来合并两者的 aspect 编织器（weaver）。

（3）利用组件语言实现的系统组件，利用 aspect 实现的 aspect 组件。

2．AOP 的程序设计步骤

AOP 应用程序包括以下三个主要的开发步骤。

（1）将系统需求进行功能性分解，区分出普通关注点以及横切关注点，确定哪些功能是组件语言必须实现的，哪些功能可以以 aspect 的形式动态加入到系统组件中。

（2）单独完成每一个关注点的编码和实现，构造系统组件和系统 aspect。这里的系统组件，是实现该系统的基本模块，对 OOP 语言，这些组件可以是类；对于过程化程序设计语言，这些组件可以是各种函数和 API。系统 aspect 是指用 AOP 语言实现的将横切关注点封装成的独立的模块单元。

（3）用联结器指定的重组规则，将组件代码和 aspect 代码进行组合，形成最终系统。为达到此目的，应用程序需要利用或创造一种专门指定规则的语言，用它来组合不同应用程序片断。这种用来指定联结规则的语言可以是一种已有编程语言的扩展，也可以是一种完全不同的全新语言。将以上过程用图 18-3 的形式来表示，该图中将系统需求看作一束光线，需求光束通过可标识关注点的棱镜将每个关注点区分开，形成单独关注点的实现，最后通过另一个 Weaver 棱镜将这些关注点整合，形成最终的应用程序。

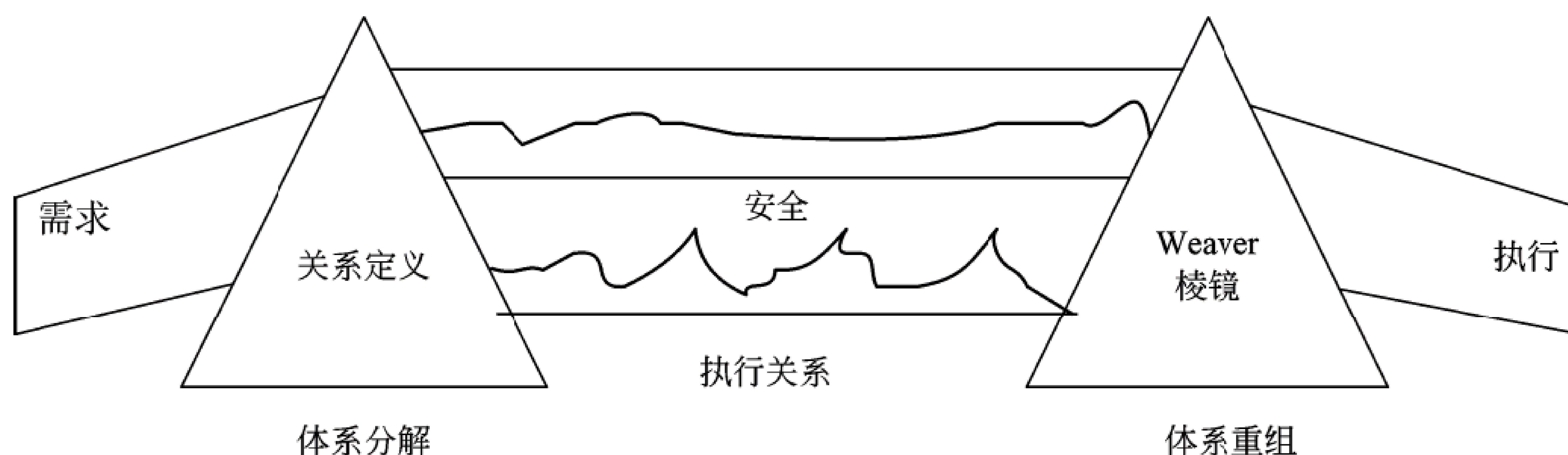


图 18-3 AOP 系统开发过程示意图

18.1.6 AOP 的优势

面向方面的技术具有很多潜在的优势，它为在系统中详细指定并封装横切点提供了方法。随着它们的发展，允许我们更好地进行系统维护。AOP 还将使我们对现存系统以一种有组织的方式增加新的特点。表达及结构方面的提高允许我们保持系统运行更长的时间，并且不会带来完全改写的开销就可以增量地对其维护。

AOP 还是质量专业人员工具箱的利器。使用 AOP 语言，可以自动测试应用程序代码而不会对代码带来干扰。这将消除可能的代码错误。

在理解 AOSD（方面面向软件设计）全部潜能中我们还处于一个初始阶段。显然，这项技术为保证未来的探索与实验提供了足够多的优点。距离每日使用 AOP 语言进行应用程序开发还有多远？这取决于我们向谁询问这个问题。

现在已经看到了一些优点，下面来看一些关于 AOSD 的风险，以及将其引入软件开发主流所需要的东西。

1．质量及风险

基于从质量的角度所做出的考察以及所完成的对 AspectJ 的探索，我已经看到了伴随着优点所带来的潜在风险。下面将讨论三个问题，以说明随着 AOSD 越来越普遍所带来的我们需要面对的关于质量的问题。

（1）如何修正我们的过程来适合 AOP。最有效的探测软件缺陷的技术之一是通过代码检查及复审。在复审中，一组程序员评论代码，以决定其是否满足了需求。在面向对象程序设计中，可以对类或一组相关类进行复审，并进行推论。可以查看代码并确定其是否正确处理了意料之外的事件，是否具有逻辑上的缺陷等。在 OO 系统中，每个类完全封装了特定概念的数据以及行为。

但是，在 AOP 中，仅仅通过代码查看，我们不再能够进行推论。我们并不知道代码是否被来自某些方面的通知所增长，或是完全被这种通知所取代。为了能够对应用程序代码做推断，我们必须能够查看来自每个类的代码，以及能够影响这个类行为的任何一个方面的代码。但是，也许这些方面还没有被编写出来。如果这样，那么当我们孤立

地考虑它的时候，我们实际上能够对这个应用程序类的代码行为理解多少呢？

实际上，考虑 AOP 代码正确性的方法与我们考虑面向对象的程序设计代码是相反。在 OOP 中，我们自内向外：我们考虑一个类，对它的上下文环境做假设，然后通过孤立的以及按照它如何与其他类交互的两种方式推断它的正确性。在 AOP 中，需要从外向内来看，并确定在可能的连接点上每个方面的效果。确定如何才能正确推断 AOP，以及开发适宜的技术和工具来帮助我们是一个值得研究的领域。

（2）测试工具以及技术的开发，特别是单元测试。由于代码可以被某一个方面所改变，当一个可以完美运行的单元测试的类被集成入一个 AOP 系统中时，可能会出现完全不同的运行状况。下面的例子说明了这一点。

例如，堆栈是一种数据结构，被用来以后进先出的模式增加或移除条目。如果向堆栈压入数据 2、4、6，然后弹出栈两次，将按照顺序得到 6 与 4。可以很直观地写出对堆栈类的单元测试，并能够很好地保证实现是正确的。但当用 AspectJ 实现了一个简单的改变——对每个条目做增一操作时，向栈上压入 2、4、6，然后从栈顶弹出两个元素。单元测试的代码并没有改变，但是行为改变了。不再是 6 和 4，而是变成了 7 和 5。

这是一个很小的例子，在真实环境中不太可能发生，但是它显示了一个恶意的程序员可以很容易地导致许多损害。即使我们忽略这种恶意的程序员，由于我们所做出的改变存在着许多副作用，许多错误仍然可能发生。要保证一个为非常有效推断所实现的方面不会对现存程序功能带来多余的效果是非常困难的。

（3）测试过程本身。一旦我们有了一组工具及技术，如何修改我们的测试过程以有效地使用它们并且能够支持我们整体的开发目标？虽然这个问题也许并不是一个主要的问题，我仍然相信在我们可以真正地对采用方面所构件的软件进行很好的测试以前需要解决它。

对 AOSD 采用的其他障碍

质量问题也许是对 AOSD 方法采用的最大阻碍，但是它们并不是唯一的。AOSD 是一种新的范例。正如其他范例一样，当它们刚刚出现时（例如面向对象的软件开发），由于所包含的学习曲线，需要经历一段时间才能被广泛采用。首先，我们需要学习基本技术以及结构，然后是高级技术，再然后是如何更好地应用技术以及什么时候它们才是最适合的。

工具在 AOP 中具有很重要的地位。除了编译器以及编辑器之外，我们需要能够帮助我们推断系统，确定潜在横切关注点，以及能够帮助我们对所存在的方面进行测试的工具。例如在 UML 中描述方面，我们的工具必须发展以支持这些方法。

此外，其他类似于 AOP 的范例也正在出现中。例如，关注点范例的多维分离，已在 IBM 研究院处于发展中（http://www.alphaworks.ibm.com/tech/hyperj）。任何对新范例的使用都是有风险的，直至对你所使用语言的标准实现被建立起来。例如，AspectJ 是一个仍然发展的 AOP 实现。风险是，也许你开发了合并入横切点的软件，你的实现方案

或者将不再被支持，或者要做很大的改动。

2．向构建软件的更好方法前进

很显然，我们具有一种能够为使 AOP 对日常应用可行化而开发工具与过程的方法。但是，这里讨论的任何问题不代表不可克服的困难。当为 AOP 开发出经得起考验的一组工具与流程时，可以找到比今天所做的更好的构建软件的方法。

正如 Barry M. Boehm 所述的关于敏捷流程，我们必须小心的采用 AOP。无论是作为早期应用者或是等待这种技术成为主流，都需要确保软件投资者在今天或是未来能够提供可接受的回报，这是非常好的商业判断力。

18.1.7 当前的 AOP 技术

当前，各种 AOP 技术层出不穷，其中相当成熟完善适用于商业开发的 AOP 技术主要包括 AspectJ、AspectWerkz、JBoss AOP 和 Spring AOP，这些皆适合用于商业开发中的开源项目。AOP 是一种概念，不同的技术可以有不同的实现。

在语法方面，AspectWerkz、JBoss AOP 和 Spring AOP 都在没有改变 Java 语言语法的情况下加入了方面语义，而 AspectJ 则对 Java 语言进行了扩展。

在声明方式方面，AspectJ 在代码中对方面进行声明。AspectWerkz 和 JBossAOP 支持用元数据对 Java 代码进行注释，或者在独立的 XML 文件中对方面进行声明。在 SpringAOP 中，则完全用 XML 对方面进行声明，比起 JBossAOP 和 AspectWerkz，SpringAOP 提供了更加精细的配置。

在性能方面，AspectJ 通过编译时对目标二进制类的增强获得面向方面能力，所以在编译时会带来开销，运行时可获得更快的速度。JBossAOP 和 SpringAOP 基于拦截技术则在运行时有更多的工作要做，对比之下，AspectJ 的构建时开销最多，AspectWerkz 次之，JBossAOP 再次，SpringAOP 没有构建时开销。

下面将重点介绍 AspectJ 和 SpringAOP 的概念构造与实践。

18.2 AspectJ

18.2.1 AspectJ 概述

AspectJ 既是一个语言规范，又是一个 AOP 语言实现。语言规范部分定义了多种语言构造以及它们支持面向方面范型的方式；语言实现部分则提供了编译、调试及从代码生成文档的工具。

AspectJ 的语言构造是从 Java 语言中扩展而来的，因此所有合法的 Java 程序也都是合法的 AspectJ 程序。AspectJ 编译器生成的是符合 Java 字节码规范的.class 文件，这使得所有符合规范的 Java 虚拟机都可以解释、执行其所生成的代码。通过选择 Java 为基

础语言，AspectJ 继承了 Java 的所有优点并使 Java 程序员能够比较容易地上手。

AspectJ 还提供了许多有用的工具。它有一个方面编织器（以编译器的方式）、一个调试器、文档生成工具以及一个独立的可用来以可视化的方式观察通知是如何切入系统各部分的方面浏览器。另外，AspectJ 还提供了与流行 IDE 的集成，如 Sun 公司的 Forte、Borland 公司的 JBuilder 以及 Emacs 等，这使得 AspectJ 成为一个很有用的 AOP 实现，特别是对 Java 开发者而言。

18.2.2　AspectJ 语言概念和构造

1．连接点

连接点是 AspectJ 中的一个重要概念，它是程序执行过程中明确定义的点。连接点可能定义在方法调用、条件检测、循环的开始或是赋值动作处。连接点有一个与之相关联的上下文。例如，一个方法调用连接点的上下文可能会包含一个目标对象及调用参数等。

虽然程序执行过程中所有可以确认的点都可以是连接点，但并不是每个点都是有用的。在 AspectJ 中，有下列可用的连接点。

（1）方法的调用（call）和执行（execution）。

（2）构造器（constructor）的调用和执行。

（3）对属性（field）的读/写访问。

（4）异常处理的执行。

（5）对象和类的初始化执行。

AspectJ 中没有提供在像 if 条件检查或 for 循环这样细粒度语言构造上的连接点。

2．切入点

切入点是用来指明所需连接点的程序构造，可以用它来指明一系列的连接点。同时，它还可以为在连接点上执行的通知提供上下文信息。例如：

```
pointcut callSayMessage().call(public static void HelloWorld.say*(…));
```

其中，pointcut 关键字表明其后是一个命名的切入点的声明。接着，callSayMessagc() 是切入点的名字，这与方法声明类似。其后的空括号表明此切入点不需要上下文信息。

再往后，call（public static void HelloWorld.say*（…））捕获所需的连接点。call 表明此切入点捕获对指定方法的调用，而不是方法的执行或是别的什么。public static void HelloWorld.say*（…）是将会产生影响的方法的签名。void 是说所捕获的方法必须要有一个 void 返回类型。HelloWorld.say*指明将要捕获的方法的类和名字。这里，我们指定 HelloWorld 类；say*使用通配符，来说明要捕获的方法应以 say 开始。最后，（…）指明了将要捕获的方法的参数列表。这里使用了“…”，表示任何形式的参数列表都在考虑范围之内。

现在，你已经知道了如何指定切入点来捕获连接点，下面再来看一下其他的切入点类型。

1）方法调用和构造器调用切入点

方法调用和构造器切入点捕获执行中准备好了方法参数后而尚未执行方法本身时的那个点。它们的形式是 ca11（方法或是构造器的签名）。

2）方法执行和构造器执行切入点

方法执行和构造器执行切入点捕获方法的执行，与调用切入点相比，执行切入点体现在方法和构造器本身。其形式为 execution（方法或是构造器的签名）。

3）属性访问切入点

属性访问切入点捕获对一个类中属性的读写访问。可以捕获所有对 System 类中的 out 属性的访问，如 System.out；也可以仅捕获读访问或写访问。举个例子来说，可以捕获对 MyClass 的属性 x 的写访问，其形式为 MyClass.x=5。读访问切入点的形式为 get（FieldSignature）；写访问切入点的形式则为 set（FieldSignature）。其中 FieldSignature 可以用与调用或执行切入点里的 McthodOrConstructor 同样的方式使用通配符。

4）异常处理切入点

异常处理切入点捕获特定类型异常处理的执行，其形式为 handler（ExceptionType-Pattern）。

5）类初始化切入点

类初始化切入点捕获类初始化部分中静态部分的执行，这里静态部分是指类定义中 Static 代码块中指定的代码。其形式为 staticinitialization（TypePattarn）。

6）基于语法结构的切入点

基于语法结构的切入点捕获一个类或方法中所有语法结构里的连接点。捕获类（包括内部类）中的语法结构连接点的切入点形式为 within（TypePattern），捕获类方法或类构造器中的语法结构连接点的切入点形式为 withincode（Method-OrConstructor-Signature）。

7）基于控制流的切入点

基于控制流的切入点捕获所有指定范围的控制流（程序的指令流）内的连接点。例如，在某个执行过程里，方法 a 调用方法 b，方法 b 就在方法 a 的控制流里。通过使用基于控制流的切入点，可以捕获由于一个方法调用而引发的所有方法调用、属性访问及异常处理等。这种类型的切入点可捕获在其控制流内的其他切入点，如果包括其自身，形式为 cflow（pointcut）；如果不包括其自身，则形式为 cflowbelow（Pointcut）。

8）基于当前对象、目标对象及参数类型的切入点

此类切入点可捕获定义在对象自身、目标对象或参数上的连接点。它是唯一可以在连接点上取得上下文的语言构造，捕获基于当前对象的连接点的切入点形式为 this（TypePattern 或 ObjecctIdentifier），捕获某个目标对象的连接点的切入点形式为 target

（TypePattern or ObjectIdentifier），基于参数的切入点形式为 args（TypePattern orObject-Identifier，...）。

9）条件测试切入点

这种切入点基于某种条件测试捕获连接点，其形式为 if（BooleanExpression）。

3．通知

通知指定当到达特定切入点处应执行的代码。AspectJ 提供了三种把通知关联到连接点的方式：before、after 及 around。before 通知在连接点的前面运行；after 通知在连接点的后面运行。对于 after 通知而言，还可以指定是在正常返回后运行还是在抛出异常后运行，或者也可以是两种情况下都运行。around 通知包在连接点的外面，并有权决定是否运行此连接点，还可以在此处修改连接点上下文环境。

4．方面

方面是 AspectJ 的模块单元，其地位就像是 Java 里的类。它把切入点和通知包在一起。和类相似，方面也可以包含方法和属性、从其他类或方面扩展以及实现接口等。与类不同的是，不能用 new 来建立一个方面实例。

AspectJ 允许在类中声明切入点，但在类中只能声明 static 的切入点。而且 AspectJ 不允许类里包含通知，只有方面可以包含通知。

方面可以标记其自身和任何切入点为抽象的（abstract）。抽象的切入点，其概念与抽象类相似，允许把细节实现推迟到派生方面里。一个具体的方面可以从抽象的方面扩展而来，它要提供抽象方面里切入点的具体定义。

18.2.3　AspectJ 实践

AspectJ 也许是已知最好的，并且应用最广泛的 AOP 实现。

图 18-4 描述了一种可以进行系统修正的方法。金融系统具有一个接口以及数个方法以更新雇员金融数据。方法名均以单词 update 开头（例如，updateFederalTaxInfo），并且每个金融更新均以雇员对象做实参。雇员个人信息也通过雇员对象，使用图 10-5 所示的方法做更新。

我们的任务是，每一次当调用任何更新函数，或是更新成功完成后，写入一个日志消息。为了简单起见，我们说我们向标准输出打印了一个日志消息。在实际系统中，我们将写入一个日志文件。下面将通过三个步骤采用 AspectJ 实现我们的解决方案。

确定在代码中需要插入日志代码的位置。这被称作在 AspectJ 中定义连接点，编写日志代码，编译新代码并将其编织入系统中。下面详细描述每一个步骤。

1．定义连接点

一个连接点是在代码中被良好定义的，我们所关注的应用程序横切的点。典型的，对每一个关注点存在着许多连接点。如果仅有一两个，通过很少的努力，就可以手工改写代码。

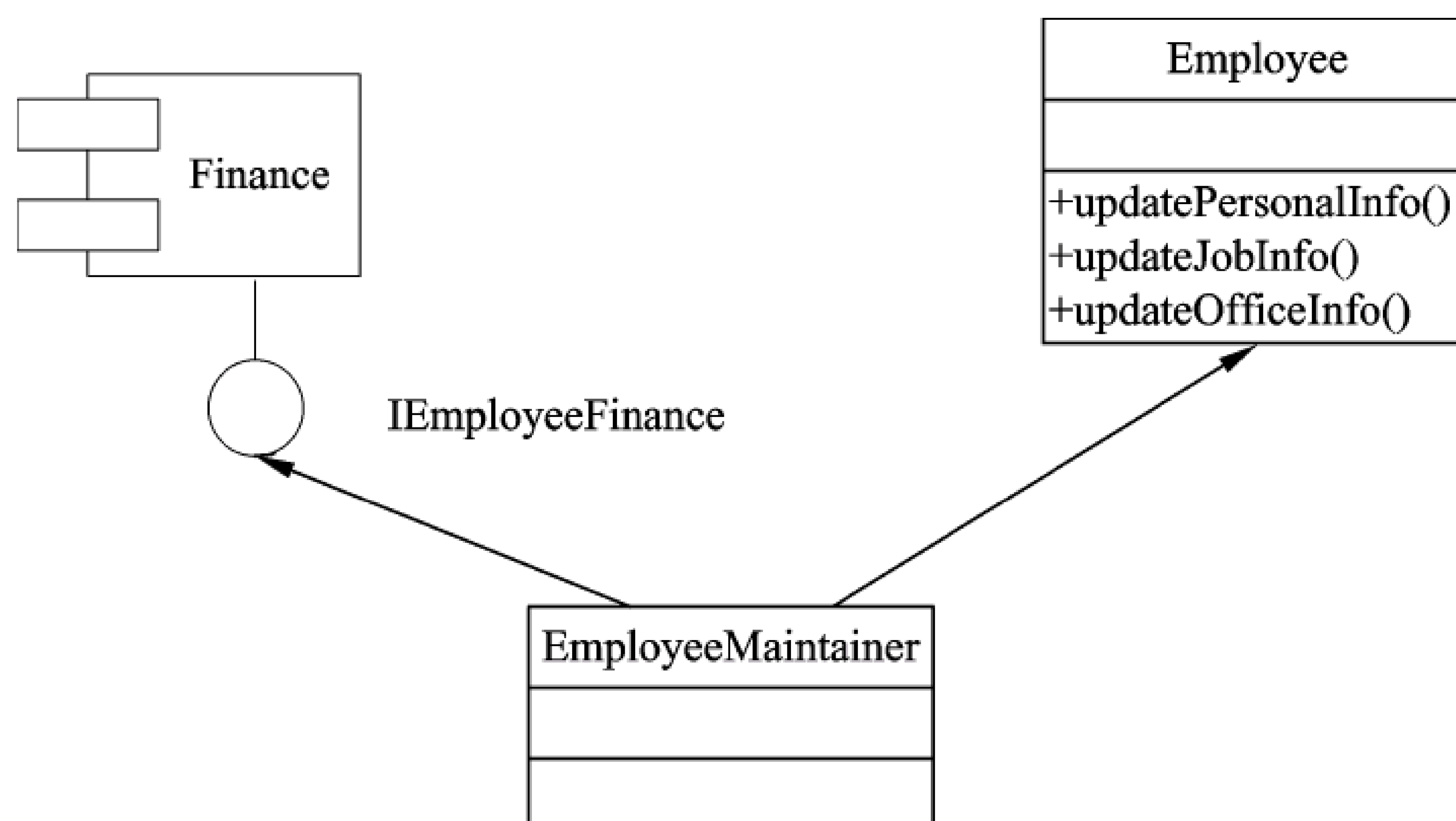


图 18-4 与更新雇员信息相关的类

在 AspectJ 中，通过将连接点分组为切点对其进行定义（AspectJ 的语法十分丰富，我们将试图在此对其进行完整的描述）。初始，定义两个切点，分别将雇员类及 IEmployeeFinance 组件中的连接点分组。下列代码定义了这两个切点。

```
pointcut employeeUpdates(Employee e):
    call(public void Employee.update*Info()) && target(e);
                                                    ...
pointcut employeeFinanceUpdates(Employee e):
    call (public void update*Info(Employee)) && args(e);
```

第一个切点称作 employeeUpdates，描述了我们调用雇员对象中以字符串 update 开头，以字符串 Info 结尾，且无实参的方法的连接点位置，它还通过 target 指示器明确指定了在雇员类中定义的方法。第二个切点 employeeFinanceUpdates，描述了所有以 update 开头，以 Info 结尾的，具有一个 Employee 类型实参的方法的调用点。合起来，这两个切点定义了所有我们关注的连接点。如果要为雇员类或 IEmployeeFinance 组件增加更多的更新方法，只要保持同样的命名规则，对它们的调用会自动被包含于切点中。这意味着当每次增加更新方法时，不需要特意地去包含日志代码。

2．编写日志代码

实现日志的代码与 Java 中其他任何方法都很相似，但是被置于一个称作方面的新风格中。方面是用来对与某一特定关注相关联的代码进行封装的一种机制。对雇主数据变更日志的方面实现如下所示。

```
public aspect EmployeeChangeLogger {
pointcut employeeUpdates(Employee e) : call( public void Employee.update*
Info()) && target(e);
pointcut employeeFinanceUpdates(Employee e) : call( public void update*
```

```
Info(Employee)) && args(e);
after(Employee e) returning : employeeUpdates(e) || employeeFinance-
Updates(e) {System.out.println("\t>Employee : " +e.getName() + " has had
a change "); System.out.println("\t>Changed by " + thisJoinPoint.getSigna-
ture()); } }
```

首先，注意到方面的结构与 Java 中的类结构很相似。典型地，方面被置于它独有的文件中，正如 Java 的类一样。虽然通常的方法是在方面代码中包含以往定义的切点，但是也可以将它们更紧密地包括在含有切点的代码中。

在切点之后，有一段与常规 Java 代码中方法相似的代码，这被称作 AspectJ 中的通知。存在着三种不同类型的通知：before、after 和 around。它们分别在连接点之前、之后，或是取代连接点而执行。还存在着许多可以使用的变种以定制你自己的通知。在我们的例子中，选择连接点返回中的更新方法之后立即运行日志。还要注意到，我们通过在冒号之后的通知头中立即分别对它们命名，以及通过逻辑“或”的方式组合两个切点。因为每个切点都有一个雇员参数，因此可以很容易地完成这项工作。

随着雇员名字，通知中的两条语句打印出了雇员信息被改变的事实。既然受到影响的雇员对象作为实参被传递给通知，那么这很容易安排。第二条语句指明了通知被执行的确切连接点，并且应用了 AspectJ 的 JoinPoint 类。 只要通知执行，仅存在一个被 thisJoinPoint 引用的关联连接点。

3．编译及测试

现在，已经编写了日志代码，接下来需要编译并将其集成入现存的系统。为了方便起见，已经实现了两个类：Employee 和 EmployeeFinance。我们还拥有一个具有主函数的简单测试类，如下所示。

```
public static void main(String[] args) {
Employee e = new Employee("Chris Smith"); // Do something to change some
of the employee's // information here.
e.updateJobInfo(); e.updateOfficeInfo(); EmployeeFinance.updateFederal-
TaxInfo(e);
EmployeeFinance.updateStateTaxInfo(e); }
```

这个代码不需要任何 AOP 实现就可以很好地运行。为了我们的例子，所有更新方法的函数体仅包含一个打印语句。当运行这个例子时，得到如下输出：

```
Updating job information
Updating office information
Updating federal tax information
Updating state tax information
```

为了将我们的方面合并入系统，我们向项目中增加了方面的源代码，并且采用

AspectJ 编译器，ajc 进行编译。编译器接受每一个方面，并建立包含通知代码的类文件。然后，在这些类文件中对适当方法的调用被编织入原始应用程序代码。 在当前 AspectJ 的发行版中，这种编织在 Java 字节码级别发生，因此不存在可以进行查阅以对最终代码进行审查的中间源文件。但是，如果你很好奇，可以对 Java 字节码进行反编译。

在开发中，我使用 Eclipse，而 AspectJ 插件保证采用正确的实参调用编译器。一旦利用 AspectJ 编译器对项目进行了编译，得到如下输出：

```
Updating job information
>Employee : Chris Smith has had a change
>Changed by void employee.Employee.updateJobInfo() Updating office
information
>Employee : Chris Smith has had a change
>Changed by void employee.Employee.updateOfficeInfo() Updating federal
tax information
>Employee : Chris Smith has had a change
>Changed by void employee.EmployeeFinance.updateFederalTaxInfo(Employee)
Updating state tax information
>Employee : Chris Smith has had a change
>Changed by void employee.EmployeeFinance.updateStateTaxInfo(Employee)
```

现在，我们知道哪个雇员信息被改变，以及改变是在哪里发生的。当然，日志可以更精细，但是基本的方法没有变化。

18.3 Spring AOP

18.3.1 Spring AOP 概述

在有很多的开放源代码和专有的 J2EE Framework 时，Spring Framework 能够脱颖而出，并且一枝独秀，我们应该相信 Spring 是独特的。Spring 定位的领域是许多其他流行的 Framework 不具有的，Spring 是全面的和模块化的，引入了方面（Aspect）提供一种新的方法来管理你的业务对象。Spring 有分层的体系结构，这意味着你能选择使用它的任何部分，它的架构仍然是内在稳定的。

Spring 的架构性，能有效地组织你的中间层对象，无论你是否选择使用了 EJB。如果你仅仅使用了 Struts 或其他的包含了 J2EE 特有 APIs 的 framework，你会发现 Spring 关注了遗留下来的问题，Spring 能消除在许多工程上对 Singleton 的过多使用。Spring 能够消除各类属性文件的定制，在 Spring 应用中大多数业务对象没有依赖于 Spring，创建的应用程序更易于单元测试。

Spring 为已建立的企业级应用提供了一个轻量级的解决方案，这个方案包括声明式

事务管理，通过 RMI 或 webservices 远程访问业务逻辑，mail 支持工具以及数据库持久化的多种选择。Spring 还提供了一个 MVC 应用框架、可以透明地把 AOP 集成到你的软件中的途径和一个优秀的异常处理体系，包括自动从 Spring 特有的异常体系中映射。

Spring 是潜在的一站式解决方案，定位于与典型应用相关的大部分基础结构。同时，Spring 也是组件化的，允许使用它的部分组件而不需牵涉其他部分。可以使用 Bean 容器，在前台展现层使用 Struts；还可以只使用 Hibernate 集成部分或是 JDBC 抽象层。Spring 是无侵入性的，意味着根据实际使用的范围，应用对框架的依赖几乎没有或是绝对最小化的。

Spring 包含许多功能和特性，并被很好地组织在图 18-5 所示的 7 个模块中。

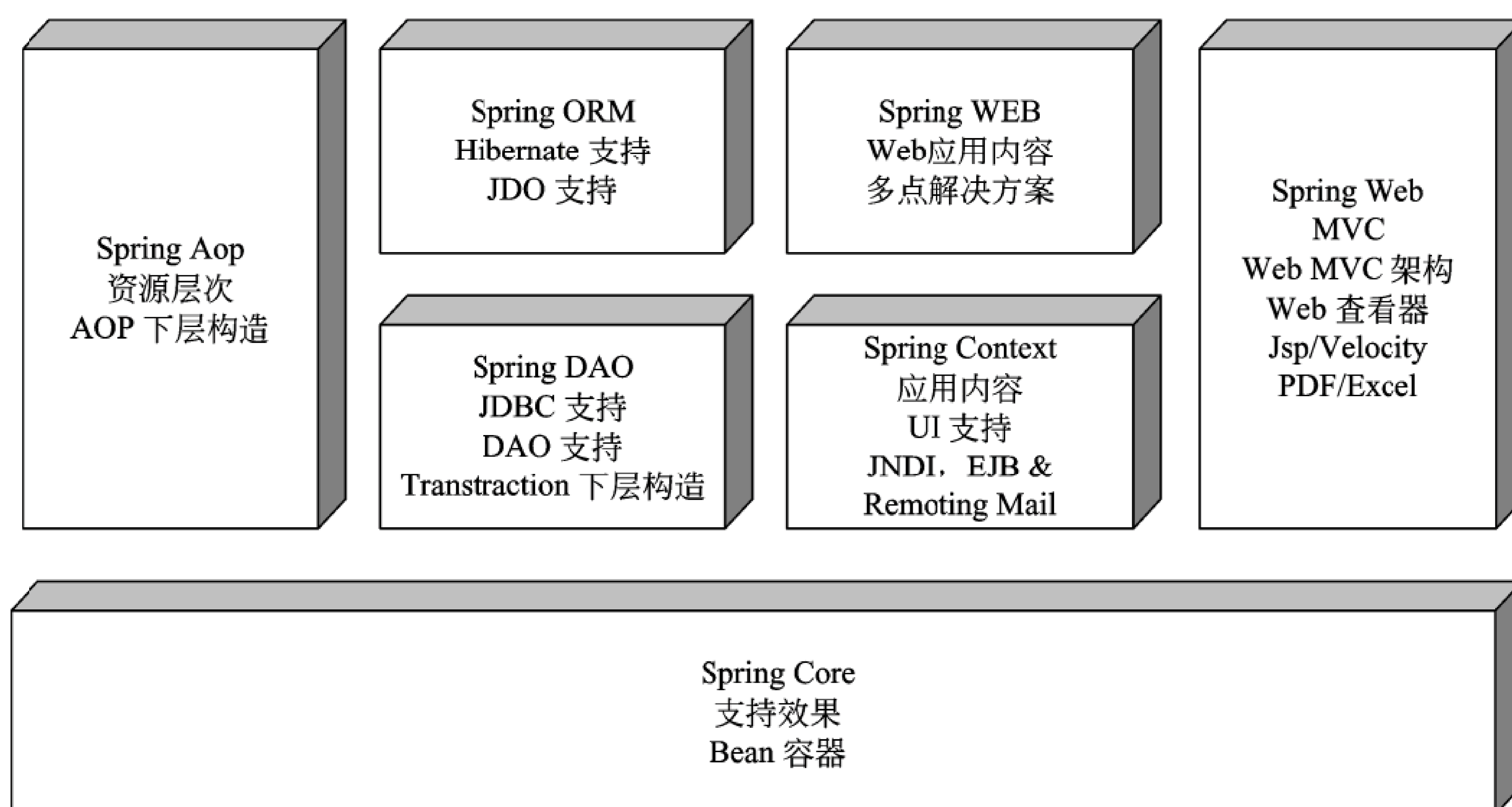


图 18-5　Spring 框架图

Core 包是框架的基础部分，并提供依赖注入特性来管理 Bean 容器功能。这里的基础概念是 BeanFactory，它提供 Factory 模式来消除对程序性单例的需要，并允许从程序逻辑中分离出依赖关系的配置和描述。

构建于 Beans 包上的 Context 包，提供了一种框架式的 Bean 访问方式，有些像 JNDI 注册。Context 包的特性得自 Beans 包，并添加了文本消息的发送，通过资源串、事件传播、资源装载的方式和 Context 的透明创建，如通过 Servlet 容器。

DAO 包提供了 JDBC 的抽象层，它可消除冗长的 JDBC 编码和解析数据库厂商特有的错误代码。该包也提供了一种方法实现编程性和声明性事务管理，不仅仅是针对实现特定接口的类，而且对所有的 POJO。

ORM 包为流行的关系——对象映射 APIs 提供了集成层，包括 JDO、Hibernate 和

iBatis。通过 ORM 包，可与所有 Spring 提供的其他特性相结合来使用这些对象/关系映射，如前边提到的简单声明性事务管理。

Spring 的 AOP 包提供与 AOP 联盟兼容的面向方面编程实现，允许定义，如方法拦截器和切点，来干净地给从逻辑上说应该被分离的功能实现代码解析。使用源码级的元数据功能，可将各种行为信息合并到你的代码中。

Spring 的 Web 包提供了基本的面向 Web 的综合特性，如 Multipart 功能，使用 Servlet 监听器的 Context 的初始化和面向 Web 的 application Context。当与 WebWork 或 Struts 一起使用 Spring 时，这个包使 Spring 可与其他框架结合。

Spring 的 Web MVC 包提供了面向 Web 应用的 Model-View-Controller 实现。Spring 的 MVC 实现不仅仅是一种实现，它提供了一种 domain model 代码和 web form 的清晰分离，这使用户可使用 Spring 框架的所有其他特性，如校验。

18.3.2 Spring 语言概念和构造

前面提到，AOP 提供从另一个角度来考虑程序结构以完善面向对象编程。面向对象将应用程序分解成各个层次的对象，而 AOP 将程序分解成各个方面或者说关注点。这使得可以模块化诸如事务管理等这些横切多个对象的关注点，称作横切关注点。

Spring 的一个关键组件就是 AOP 框架。Spring IoC 容器（BeanFactory 和 Application-Context）并不依赖于 AOP，这意味着如果不需要，可以不使用 AOP。AOP 完善了 Spring IoC，使之成为一个有效的中间件解决方案。

1．AOP 在 Spring 中的使用

（1）提供声明式企业服务，特别是作为 EJB 声明式服务的替代品。这些服务中最重要的是声明式事务管理，这个服务建立在 Spring 的事务管理抽象之上。

（2）允许用户实现自定义的方面，用 AOP 完善他们的 OOP 的使用。这样，可以把 Spring AOP 看作是对 Spring 的补充，它使得 Spring 不需要 EJB 就能提供声明式事务管理；或者使用 Spring AOP 框架的全部功能来实现自定义的方面。

2．Spring AOP 的功能

Spring AOP 用纯 Java 实现，不需要特别的编译过程，区别于 AspectJ 的实现。Spring AOP 不需要控制类装载器，因此适用于 J2EE Web 容器或应用服务器。

Spring 目前支持拦截方法调用。成员变量拦截器没有实现，虽然加入成员变量拦截器支持并不破坏 Spring AOP 核心 API。Spring 提供代表切入点或各种通知类型的类。Spring 使用术语 advisor 来表示代表方面的对象，它包含一个通知和一个指定特定连接点的切入点。各种通知类型有 MethodInterceptor，来自 AOP 联盟的拦截器 APD 和定义在 org.springframework.aop 包中的通知接口。所有通知必须实现 org.aopalliance.aop.Advice 标签接口。取出就可使用的通知有 MethodInterceptor、ThrowsAdvice、BeforeAdvice 和 AfterReturningAdvice。

Spring 实现 AOP 的途径不同于其他大部分 AOP 框架，它的目标不是提供及其完善的 AOP 实现（虽然 Spring AOP 非常强大）；而是提供一个和 Spring IoC 紧密整合的 AOP 实现，帮助解决企业应用中的常见问题。因此，例如 Spring AOP 的功能通常是和 Spring IoC 容器联合使用的。AOP 通知是用普通的 bean 定义语法来定义的（虽然可以使用 autoproxying 功能）。通知和切入点本身由 Spring IoC 管理，这是一个重要的其他 AOP 实现的区别。有些是使用 Spring AOP 无法容易或高效地实现，例如通知非常细粒度的对象。这种情况 AspectJ 可能是最合适的选择。但是，我们的经验是 Spring 针对 J2EE 应用中大部分能用 AOP 解决的问题提供了一个优秀的解决方案。

3．Spring AOP 的重要概念

前面已经提到了 AOP 的重要概念，下面介绍在 Spring 中的定义和实现。

（1）方面：一个关注点的模块化，这个关注点的实现可能横切另外多个对象。事务管理是 J2EE 应用中一个很好的横切关注点例子。方面用 Spring 的 Advisor 或拦截器实现。

（2）连接点：程序执行过程中明确的点，如方法的调用或特定的异常被抛出。

（3）通知：在特定的连接点，AOP 框架执行的动作。各种类型的通知包括 around、before 和 throws 通知。许多 AOP 框架包括 Spring 都是以拦截器做通知模型，维护一个“围绕”连接点的拦截器链。

（4）切入点：指定一个通知将被引发的一系列连接点的集合。AOP 框架必须允许开发者指定切入点。例如，使用正则表达式。

（5）引入：添加方法或字段到被通知的类。Spring 允许引入新的接口到任何被通知的对象。例如，可以使用一个引入使任何对象实现 IsModified 接口，来简化缓存。

（6）目标对象：包含连接点的对象。也被称作被通知或被代理对象。

（7）AOP 代理：AOP 框架创建的对象，包含通知。在 Spring 中，AOP 代理可以是 JDK 动态代理或者 CGLIB 代理。

（8）织入：组装方面来创建一个被通知对象。这可以在编译时完成（例如使用 AspectJ 编译器），也可以在运行时完成。Spring 和其他纯 Java AOP 框架一样，在运行时完成织入。

特别指出，Spring 默认使用 JDK 动态代理实现 AOP 代理。这使得任何接口或接口的集合能够被代理。Spring 也可以是 CGLIB 代理。这可以代理类，而不是接口。如果业务对象没有实现一个接口，CGLIB 被默认使用。但是，作为一针对接口编程而不是类编程的良好实践，业务对象通常实现一个或多个业务接口。

前面提到横切关注点是 AOP 中的重要因素，使之独立于 OO 的层次选定目标，横切点到系统的切入点理所当然是构成系统的结构要素。下面看看 Spring 是如何处理切入点这个重要因素的。

Spring 的切入点模型能够使切入点独立于通知类型被重用，同样的切入点有可能接

受不同的通知。org.springframework.aop.Pointcut 接口是重要的接口，用来指定通知到特定的类和方法目标。完整的接口定义如下：

```
public interface Pointcut{
    CLassFilter getClassFilter();
    MethodMatcher getMethodMatcher();
}
```

将 Pointcut 接口分成两个部分有利于重用类和方法的匹配部分，并且组合细粒度的操作（如和另一个方法匹配器执行一个“并”的操作）。ClassFilter 接口被用来将切入点限制到一个给定的目标类的集合。如果 matches 永远返回 true，所有的目标类都将被匹配。

```
public interface ClassFilter{
boolean matches(Class clazz);
}
```

MethodMatcher 接口通常更加重要。完整的接口如下：

```
public interface MethodMatcher{
boolean matches(Method m, Class targetClass);
boolean isRuntime();
    boolean matches(Method m, Class targetClass, Object[] args);
}
```

matches（Method, Class）方法被用来测试这个切入点是否匹配目标类的给定方法。这个测试可以在 AOP 代理创建的时候执行，避免在所有方法调用时都需要进行测试。如果两个参数的匹配方法对某个方法返回 true，并且 MethodMatcher 的 isRuntime()也返回 true，那么三个参数的匹配方法将在每次方法调用时被调用。这使得切入点能够在目标通知被执行之前立即查看传递给方法调用的参数。

大部分 MethodMatcher 都是静态的，意味着 isRuntime()方法返回 false。这种情况下，三个参数的匹配方法永远不会被调用。如果可能，尽量使切入点是静态的，使当 AOP 代理被创建时，AOP 框架能够缓存切入点的测试结果。当然，目前的技术只实现了方面静态织入，无法动态地在运行状态下组合方面。

18.3.3 Spring AOP 应用

Spring AOP 是 Spring 框架的重要组成部分，它实现了 AOP 联盟约定的接口。Spring AOP 是由纯 Java 开发完成的，它实现了方法级别的连接点，而在 J2EE 应用中，AOP 拦截到方法级的操作已经足够了。由于 OOP 倡导的是基于 setter/getter 的方法访问，而非

直接访问域，所以 Spring 仅仅提供方法级的连接点。为了使控制反转（IoC）很方便地使用健壮、灵活的企业服务，需要 Spring AOP 来实现，因为它在运行时才创建 Advice 对象。下面讨论使用 Spring AOP 松散耦合的几种方式。

1．创建通知

为实现 AOP，开发者需要开发 AOP 通知（Advice）。AOP 通知包含了方面（Aspect）的业务逻辑。当创建一个 Advice 对象时，就编写了实现横切（cross-cutting）功能的代码。Spring 的连接点是用方法拦截器实现的，这就意味着编写的 Spring AOP 通知将在方法调用的不同点织入程序中。由于在调用一个方法时有几个不同的时间点，Spring 可以在不同的时间点织入程序。

Spring AOP 中，提供了如下 4 种通知的接口。

（1）MethodBeforeAdvice：用于在目标方法调用前触发。

（2）AfterReturningAdvice：用于在目标方法调用后触发。

（3）ThrowsAdvice：用于在目标方法抛出异常时触发。

（4）MethodInterceptor：用于实现 Around 通知（Advice），在目标方法执行的前后触发。

如果要实现相应的功能，则需要实现上述接口。例如，实现 Before 通知（Advice）需要实现方法 void before（Method method，Object[] args，Object target）；实现 After 通知（Advice）需要实现方法 void afterReturning（Method method，Object [] args，Object target）。

2．在 Spring 中定义切入点

在不能明确调用方法时，通知就很不实用。切入点则可以决定特定的类、特定的方法是否匹配特定的标准。如果匹配，则通知将应用到此方法上。Spring 切入点允许用很灵活的方式将通知组织进我们的类中。Spring 中的切入点框架的核心是 Pointcut 接口，此接口允许定义织入通知中的类和方法。许多方面就是通过一系列的通知和切入点组合来定义的。

在 Spring 中，一个 advisor 就是一个方面的完整的模块化表示。Spring 提供了 PointcutAdvisor 接口把通知和切入点组合成一个对象。Spring 中很多内建的切入点都有对应的 PointcutAdvisor，因此可以很方便地在一个地方管理切入点和通知。Spring 中的切入点分为两类：静态和动态。因为静态切入点的性能要优于动态切入点，所以优先考虑使用静态切入点。Spring 为我们提供创建静态切入点很实用的类 StaticMethodMather-Pointcut，在这个类中，只需要关心 setMappedName 和 setMappedNams 方法，可以使用具体的类名，也可以使用通配符。例如，设置 mappedName 属性为 set*，则匹配所有的 set 方法。Spring 还提供了通过正则表达式来创建静态切入点的实用类 RegexpMethod-Pointcut。通过使用 Perl 样式的正则表达式来定义感兴趣的方法。当切入点需要用运行时参数值来执行通知时，则使用动态切入点。Spring 提供了一个内建的动态切入点

ControlFlowPointcut，此切入点匹配基于当前线程的调用堆栈。只有在当前线程运行时找到特定的类和特定的方法才返回 true，使用动态切入点有很大的性能损耗。大多数的切入点可以静态确定，我们很少有机会创建动态切入点。为了增加切入点的可重用性，Spring 提供了切入点上的集合操作——交集和并集。

3. 用 ProxyFactoryBean 创建 AOP 代理

ProxyFactoryBean 和其他 Spring 的 FactoryBean 实现一样，引入一个间接的层次。如果定义一个名字为 myfactory 的 ProxyFactoryBean，引用 myfactory 的对象所看到的不是 ProxyFactoryBean 实例本身，而是由实现 ProxyFactoryBean 的类的 getObject ()方法所创建的对象。这个方法将创建一个包装了目标对象的 AOP 代理。使用 ProxyFactoryBean 或者其他 IoC 可知的类来创建 AOP 代理最重要的一个优点是 IoC 可以管理通知和切入点。这是一个非常强大的功能，能够实现其他 AOP 框架很难实现的特定的方法。例如，一个通知本身可以引用应用对象（除了目标对象，它在任何 AOP 框架中都可以引用应用对象），这完全得益于依赖注入所提供的可插入性。通常，不需要 ProxyFactoryBean 的全部功能，因为我们常常只对一个方面感兴趣。例如，事务管理。当我们仅仅对一个特定的方面感兴趣时，可以使用许多便利的工厂来创建 AOP 代理，如 TransactionProxyFactoryBean。

4. 自动代理

在应用规模比较小，只有很少类需要被通知时，ProxyFactoryBean 可以很好地工作。当有许多类需要被通知时，创建每个代理就显得很烦琐。幸运的是，Spring 提供了使用自动通过容器来创建代理的功能。这时，只需要配置一个 Bean 来做烦琐的工作。Spring 提供了两个类实现自动代理：BeanNameAutoProxyCreator 和 DefaultAdvisorAutoProxyCreator。BeanNameAutoProxyCreator 为匹配名字的 Bean 产生代理，它可将一个或者多个方面应用在命名相似的 Bean 中。自动代理框架将自动产生代理要暴露出的接口。如果目标 Bean 没有实现任何接口，就会动态产生一个子类。而更强大的自动代理是 DefaultAdvisorAutoProxyCreator，只需要在 BeanFactory 中包含它的配置就可完成代理。这个类的奇妙之处在于它实现了 BeanPostProcessor 接口。当 Bean 定义被加载到 Spring 容器中后，DefaultAdvisorAutoProxyCreator 将搜索上下文中的 Advisor，最后它将 Advisor 应用到匹配 Advisor 切入点的 Bean 中。这个代理只对 Advisor 起作用，它需要通过 Advisor 来得到需要通知的 Bean。元数据自动代理（MetaDataAutoProxy）配置依赖于源代码属性而不是外部 XML 配置文件。这可以非常方便地将源代码和 AOP 元数据组织在同一个地方。元数据自动代理最常用的地方是用来声明事务，Spring 提供了很强的 AOP 框架来声明事务。

第 19 章　嵌入式系统设计

嵌入式软件是与硬件最为相关的软件系统。随着嵌入式设备的增长，嵌入式软件复杂度也不断增加。本章介绍了嵌入式系统的特点及针对嵌入式系统的软件设计。

19.1　嵌入式系统

19.1.1　嵌入式系统概念

1．嵌入式系统的基本概念

嵌入式系统是一种以应用为中心，以计算机技术为基础，可以适应不同应用对功能、可靠性、成本、体积和功耗等方面的要求，集可配置可裁减的软、硬件于一体的专用计算机系统。主要由嵌入式硬件平台、相关支撑硬件、嵌入式操作系统、支撑软件以及应用软件组成。

嵌入式系统具有以下特点。

（1）系统专用性强。

（2）系统实时性强。

（3）软、硬件依赖性强。

（4）处理器专用。

（5）多种技术紧密结合。

（6）系统透明性。

（7）系统资源受限。

2．嵌入式系统的实时概念

兼有实时系统的特性和嵌入式系统特性的系统称为实时嵌入式系统。它们之间的关系如图 19-1 所示。

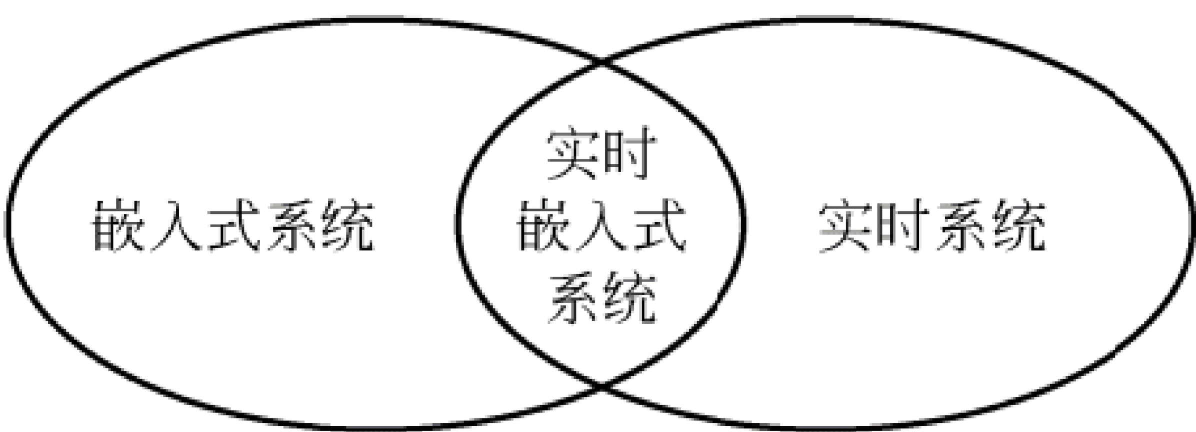


图 19-1　实时嵌入式系统

3．嵌入式系统的分类

按照嵌入方式、嵌入程度、实时性和系统的复杂程度 4 种准则可以对现有的嵌入式系统进行如下分类。

根据嵌入方式分类：整机式嵌入、部件式嵌入和芯片式嵌入。

根据嵌入程度分类：深度嵌入、中度嵌入和浅度嵌入。

根据实时性分类：实时嵌入式系统和非实时嵌入式系统。

根据系统的复杂程度分类：单微处理器嵌入式系统、组件式嵌入式系统和分布式嵌入式系统。

4．嵌入式系统的应用领域

嵌入式系统和嵌入式软件的主要应用领域如下。

（1）工业控制领域。

（2）家电领域。

（3）商业和金融领域。

（4）交通运输领域。

（5）通信领域。

（6）建筑领域。

（7）环境监测领域。

（8）医疗卫生领域。

19.1.2 嵌入式系统的基本架构

1．嵌入式系统

嵌入式系统一般由软件和硬件两个部分组成，其中嵌入式处理器、存储器和外部设备等（如图 19-2 所示）构成整个系统的硬件基础。嵌入式系统的软件部分可以分为多个层次（如图 19-3 所示），其中系统软件和支撑软件是基础，应用软件则是最能体现整个嵌入式系统的特点和功能的部分。

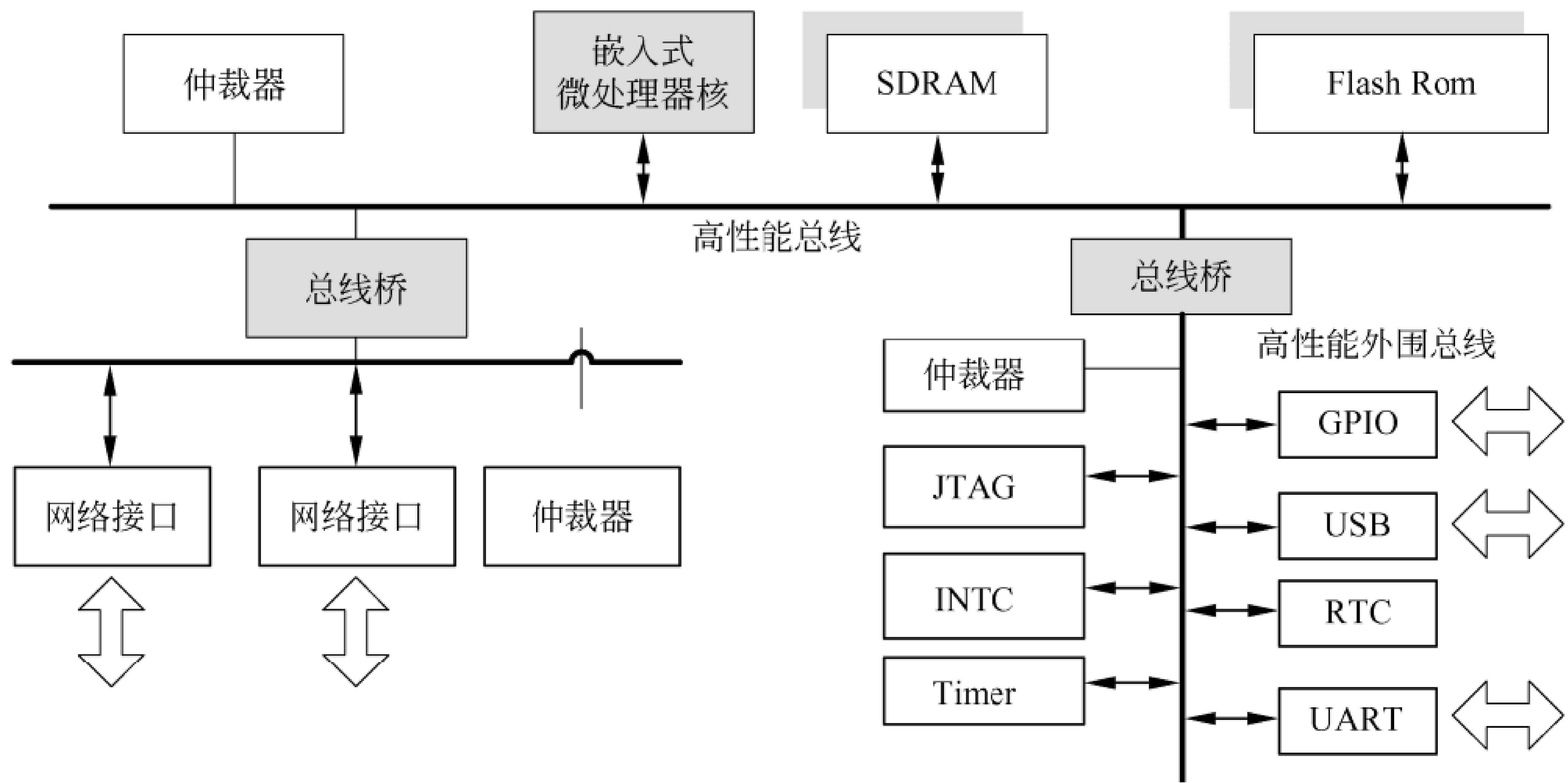


图 19-2 典型嵌入式系统硬件架构

2．硬件平台的系统架构

微处理器是整个嵌入式系统的核心，负责控制系统的执行。根据目前的使用情况，

嵌入式处理器可以分为如下几类。

（1）嵌入式微处理器。

（2）嵌入式微控制器。

（3）嵌入式数字信号处理器。

（4）嵌入式片上系统。

3．嵌入式系统的软件架构

一个完整的嵌入式软件体系如图 19-3 所示。这个体系自底向上由以下部分组成：设备驱动管理层、嵌入式操作系统、支撑软件和应用软件。

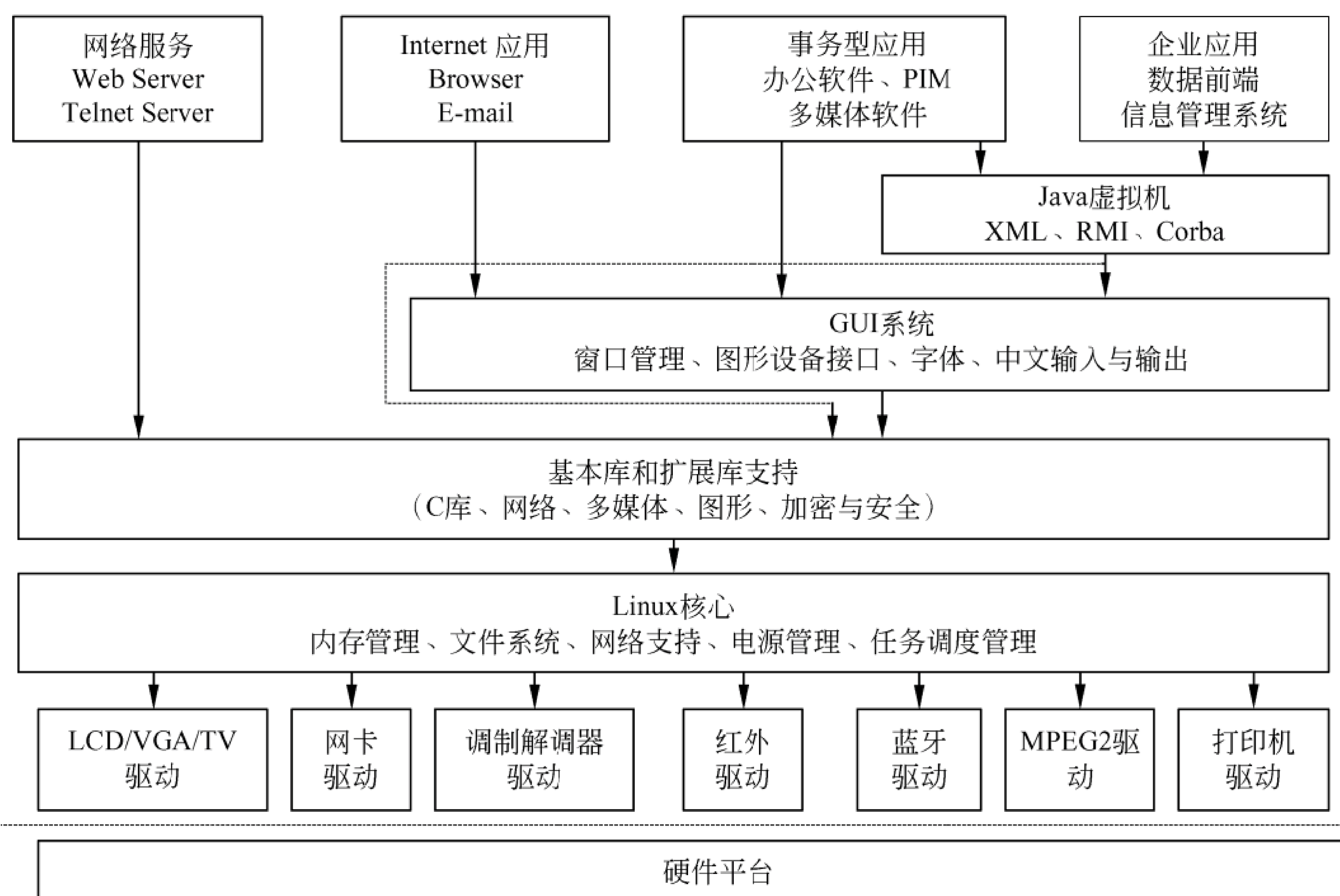


图 19-3　一种基于 Linux 的嵌入式软件架构

例如，在 Linux 核心操作系统的架构中，文件系统、网络支持等部分都以模块化的方式与核心协调工作。整个 Linux 核心的源代码树可以通过工程手段根据需要进行剪裁，从中剔除掉不必要的设备驱动程序、文件系统、语言和显示等的支持，从而在保证核心具备必须功能的前提下达到精简核心尺寸的目的。

存储方案支持固化 Flash、CF 卡、DOC/DOM/DOF 以及各种低噪音的嵌入式硬盘。在实际的开发过程中，存储方案的选择与具体的应用模式相结合，根据不同的应用模式来采用相应的存储方案。实际上，存储方案的选择就是在嵌入式 Linux 系统的可靠性、尺寸、功能、成本之间寻求最佳的平衡点。

因此，嵌入式软件需要通过裁减与组合，以适应各种应用场合和成本需要。嵌入式软件架构需要很好的可配置性和扩展性。

19.1.3 嵌入式操作系统

1．嵌入式操作系统的概念与特点

嵌入式操作系统是指运行在嵌入式计算机系统上支持嵌入式应用程序的操作系统，是用于控制和管理嵌入式系统中的硬件和软件资源、提供系统服务的软件集合。嵌入式操作系统是嵌入式软件的一个重要组成部分。

与通用操作系统相比，嵌入式操作系统主要有以下特点。

（1）微型化。

（2）代码质量高。

（3）专业化。

（4）实时性强。

（5）可裁减、可配置。

2．嵌入式操作系统的分类

从嵌入式操作系统的获得形式上，可以分为商业型和非商业型两类。

根据嵌入式操作系统的实时性，可以分为实时嵌入式操作系统和非实时嵌入式操作系统两类。

3．嵌入式操作系统的一般结构

嵌入式操作系统的一般结构如图 19-4 所示。嵌入式操作系统主要由应用程序接口、设备驱动和操作系统内核等几部分组成。

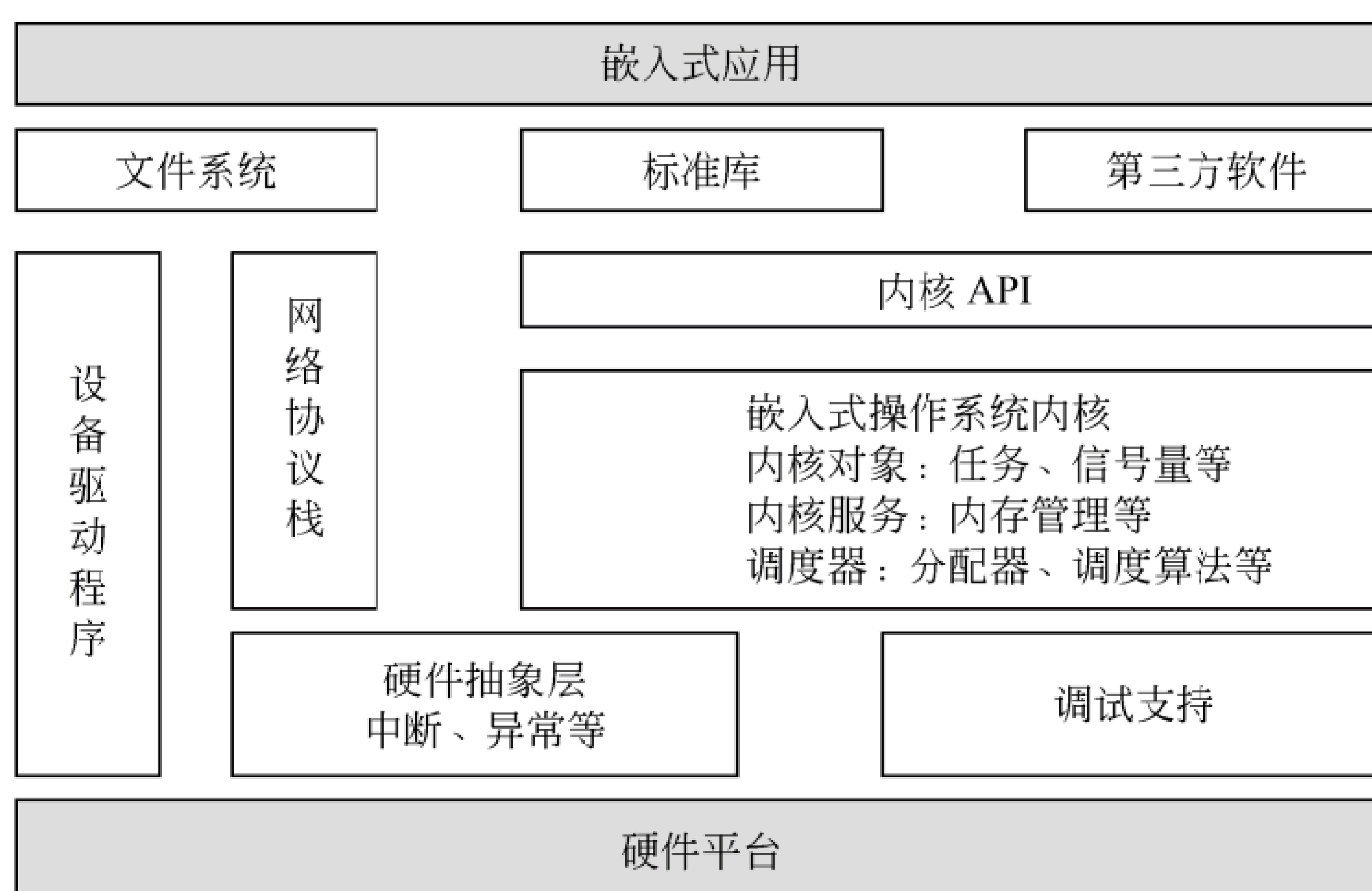


图 19-4 嵌入式操作系统的一般结构

嵌入式操作系统是一个按时序方式调度执行、管理系统资源并为应用代码提供服务的基础软件。每个嵌入式操作系统都有一个内核。另一方面，嵌入式操作系统也可以是各种模块的有机组合，包括内核、文件系统、网络协议栈和其他部件。但是，如图 19-5 所示，大多数内核都包含以下三个公共部件：调度器、内核对象和内核服务。

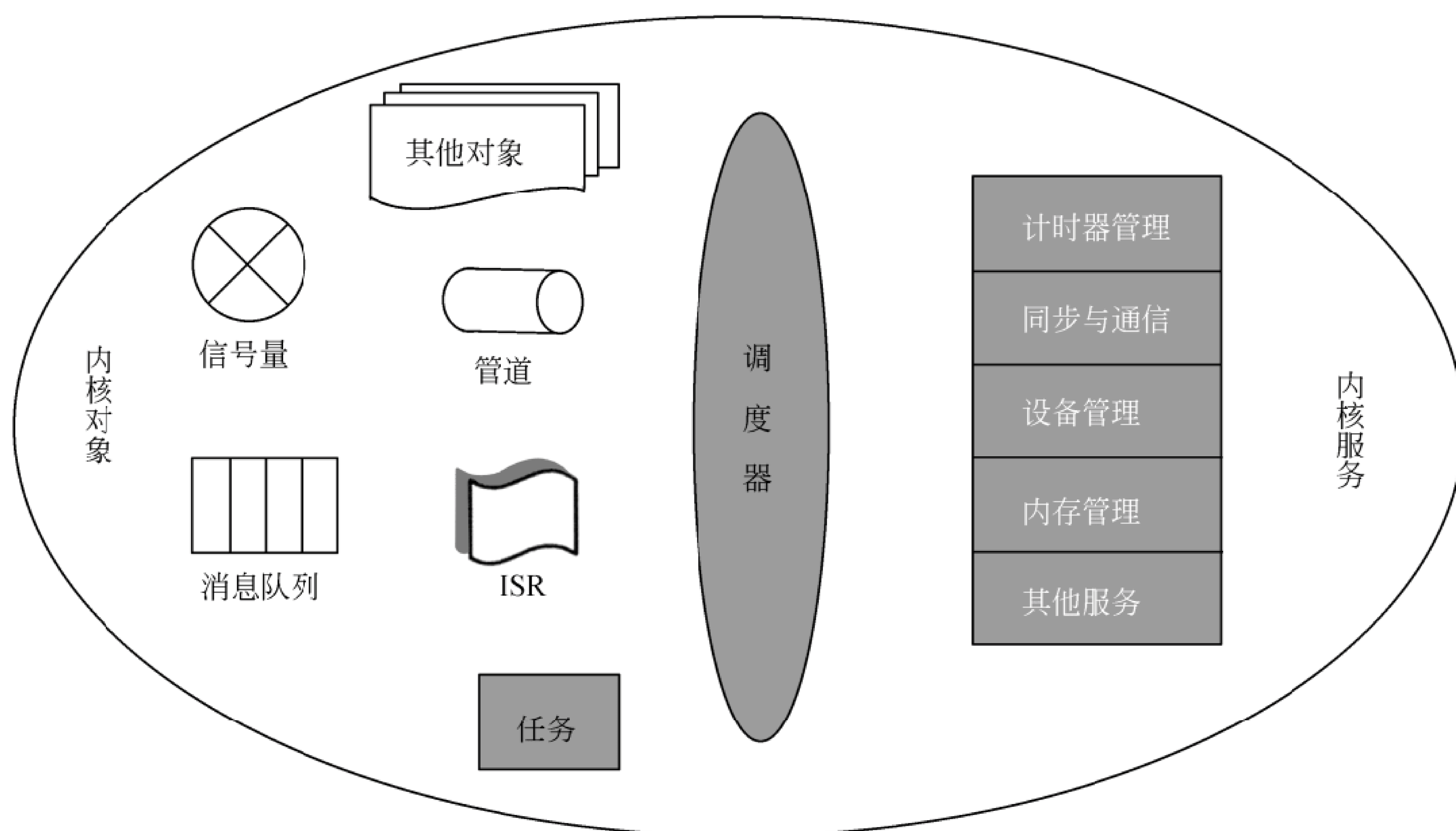


图 19-5　嵌入式操作系统的内核部件

4．嵌入式操作系统的多任务调度

1）基本概念

（1）任务。

（2）任务对象。

（3）多任务。

（4）调度器。

（5）可调度实体。

（6）上下文切换。

（7）可重入性。

（8）分发器。

（9）调度算法。

（10）优先级。

目前，大多数内核支持两种普遍的调度算法，即基于优先级的抢占调度（Preemptive Priority-Based Scheduling）和时间轮转调度算法（Round-Robin Scheduling）。

2）调度算法

（1）任务优先级分配方法。

一般地，可以采用单调执行速率调度法 RMS（Rate Monotonic Scheduling）来给任务分配优先级，执行最频繁的任务优先级最高。

（2）时间轮转调度。

基于优先级抢占式扩充时间轮转调度，对于优先级相同的任务使用时间片获得相等的 CPU 执行时间。内核在满足以下条件时，把 CPU 控制权转交给下一个就绪态的任务。

（3）任务操作。

内核提供任务管理服务，也提供一个允许开发者操作任务的系统调用。典型的任务操作有任务创建和删除、任务调度控制、任务信息获取。

5. 嵌入式操作系统的内核对象

实时嵌入式操作系统的用户可以使用内核对象来解决系统设计中的问题，如并发、同步与互斥、数据通信等。内核对象包括信号量、消息队列、管道、事件与信号等。

6. 实时嵌入式操作系统的内核服务

实时嵌入式操作系统的内核服务有异常和中断、计时器、I/O 管理。

7. 内存管理

不论嵌入式系统的类型如何，对内存系统的普遍要求是最高的内存利用率、最小的管理负载和确定的分配时间。管理内容如下。

（1）嵌入式系统中固定尺寸内存池的内存管理。

（2）阻塞与非阻塞的内存函数。

（3）硬件内存管理单元。

（4）同步与通信。

19.1.4 典型嵌入式操作系统

嵌入式操作系统分为从不同的通用操作系统发展来的通用嵌入式操作系统，如 Win CE、Linux 等，大多数是特定领域专用操作系统，如表 19-1 所示。

表 19-1 主要嵌入式操作系统

名　称	简　介
ECOS	美国 Cygnus Solutions 公司开发的源代码开放的嵌入式操作系统，使用于深度嵌入式应用。主要用在信息电器上，如数字电视、冰箱和空调等
EPOC	Psion Software 公司推出的一个 16/32 位多任务嵌入式操作系统，在移动计算设备中应用广泛，在 PDA 手机市场上占有相当的份额。目前支持 EPOC 的主要有 Ericsson、Motorola、Panasonic、Nokia 和 Psion PLC 等公司
IOS	Cisco 公司推出的一个专用嵌入式操作系统，主要用在网络交换机、路由器等网络设备上
LynxOS	Lynx Real-time Systems 开发的一个分布式、可扩展的嵌入式实时操作系统，有很高的市场占有率

续表

名　称	简　介
Nucleus	Accelerated Technology 公司开发的一个嵌入式实时操作系统，主要用在消费电子、网络设备、无线、导航、办公设备、医疗设备和控制等领域。可以向用户开放源代码，在美国有很高的市场占有率
OS-9	Microware Systems 公司开发的一个嵌入式实时操作系统，其市场占有率很高，在国外排在前十名，主要用在高科技产品中，包括消费电子产品、工业自动化、无线通信产品、医疗仪器、数字电视以及多媒体设备中。它提供了很好的安全和容错性能，与其他的嵌入式系统相比，更具灵活性
pSOS	Integrated Systems 公司研发的一个产品，是世界上最早的实时系统之一，是一个模块化的操作系统，比较适用于深度嵌入式系统中。还配有一系列的基于 pSOS 的支撑软件，这些软件包括 TCP/IP 协议栈 pNA、远程过程调用库 pRPC、文件系统管理 pHILE、ANSI C、标准库 pREPC、调试功能模块 pROBE 及信息系统实时分析工具 pMONT 等
QNX	加拿大 QNX Software Systems Europe 公司研制的一个实时、可扩展操作系统，并部分遵循 POSIX 相关标准，采用微内核结构。微内核小巧，主要提供 4 种基本服务，所有的操作系统服务都是能互相通信的用户进程。目前，支持 X86、Power PC、MIPS 和 ARM 等处理器。主要的应用领域是消费电子、电信、汽车及医疗设备等
VxWorks	美国 WindRiver 公司于 1983 年设计开发的一种嵌入式实时操作系统，是 Tornadoll 嵌入式开发环境的关键组成部分。良好的持续发展能力、高性能的内核以及友好的用户开发环境，在嵌入式实时操作系统领域逐渐占据一席之地。首先，它十分灵活，具有多达 1800 个功能强大的应用程序接口（API）。其次，适用面广，可以适用于从最简单到最复杂的产品设计。再次，可靠性高，可以用于从防抱死刹车系统到星际探索的关键任务。最后，适用性强，可以用于所有流行的 CPU 平台

以 VxWroks 为例，VxWroks 是一个运行在目标机上的高性能、可裁减的嵌入式实时操作系统。VxWroks 是专门为实时嵌入式系统设计开发的操作系统内核，为程序员提供了高效的实时多任务调度、中断管理、实时的系统资源以及实时的任务间通信。VxWroks 在各种 CPU 平台上提供了统一的编程接口和一致的运行特性，尽可能地屏蔽了不同 CPU 之间的底层差异。

VxWroks 以其良好的可靠性和卓越的实时性被广泛地应用在通信、军事、航空和航天等高精尖技术及实时性要求极高的领域中，如卫星通信、军事演习、弹道制导和飞机导航等。

VxWroks 操作系统的基本构成部件主要有以下 5 个部分：板级支持包 BSP（Board Support Package）、微内核 Wind、网络系统、文件系统和 I/O 系统。

VxWorks 系统具有高性能的微内核设计、可裁剪的运行软件、综合的网络工具、兼容 POSIX 1003.1b 标准、平台的选择、方便地移植到用户硬件上及操作系统选件等特色。

19.1.5 嵌入式数据库管理

1．嵌入式数据库管理系统概述

通常，嵌入式数据库管理系统就是在嵌入式设备上使用的数据库管理系统。由于用到嵌入式数据库管理系统的系统大多数都是移动信息设备，所以，嵌入式数据库也称为移动数据库或嵌入式移动数据库。其作用主要是解决移动计算环境下数据的管理问题，移动数据库是移动计算环境中的分布式数据库。

2．嵌入式数据库管理系统使用环境的特点

嵌入式数据库系统是一个包含嵌入式数据库管理系统在内的跨越移动通信设备、工作站或台式机以及数据服务器的综合系统。其使用环境的特点可以简单地归纳如下。

（1）设备随时移动性。

（2）网络频繁断接。

（3）网络条件多样化。

（4）通信能力不对称。

3．嵌入式数据库管理系统组成与关键技术

1）嵌入式数据库管理系统组成

一个完整的嵌入式数据库管理系统包括主数据库管理系统、同步服务器、嵌入式数据库管理系统和连接网络等几个子系统，如图 19-6 所示。

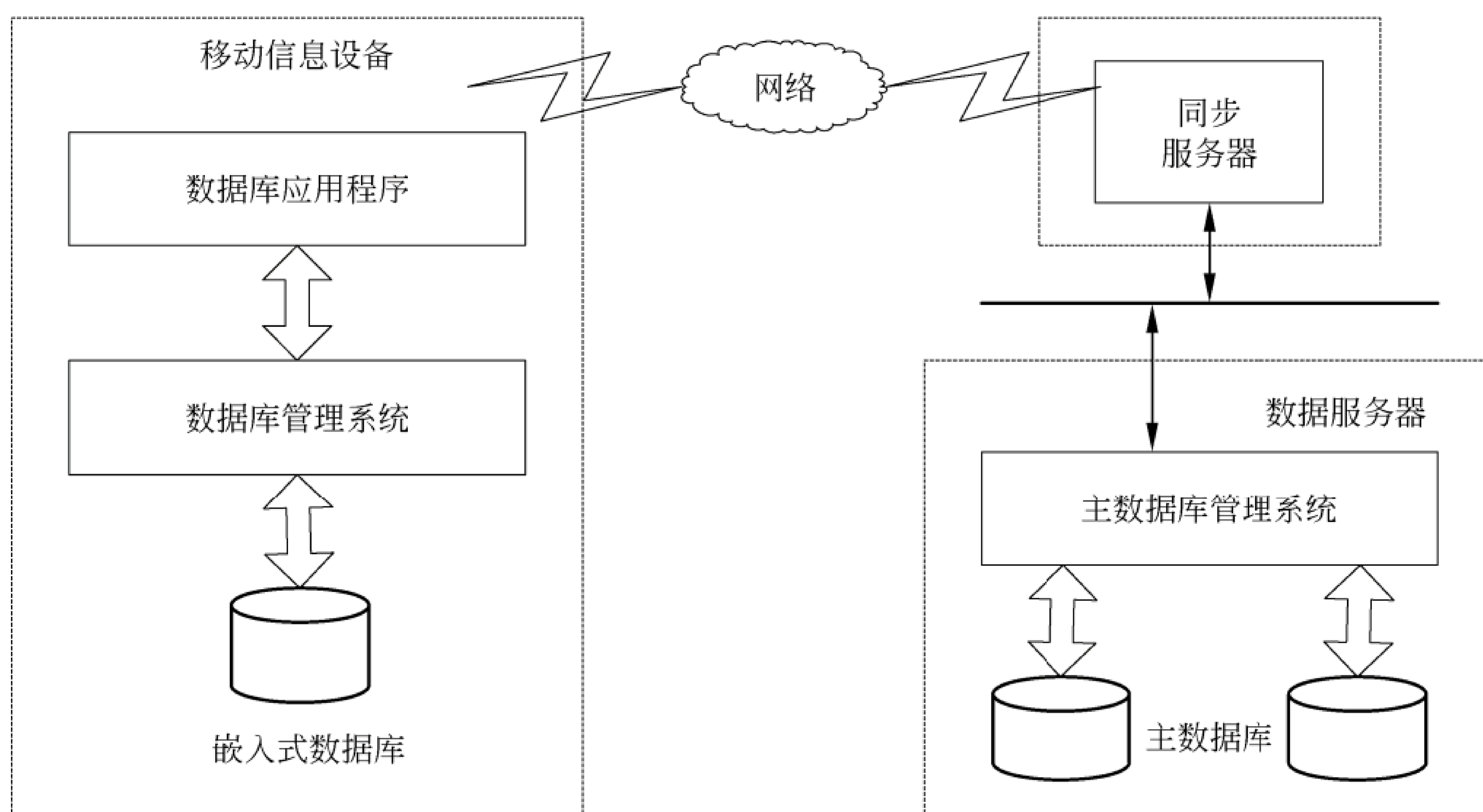


图 19-6　嵌入式数据库系统组成

2）嵌入式移动数据库在应用中的关键

嵌入式移动数据库在实际应用中必须解决好数据的一致性（复制性）、高效的事务

处理和数据的安全性等问题。

3）移动数据库管理系统的特性

由于嵌入式移动数据库管理系统在移动计算的环境下应用在嵌入式操作系统之中，是一种动态分布式数据库管理系统，其特点和功能如下。

（1）微核结构。

（2）对标准 SQL 的支持。

（3）事务管理功能。

（4）完善的数据同步机制。

（5）支持多种连接协议。

（6）完备的嵌入式数据库管理功能。

（7）支持多种嵌入式操作系统。

还应考虑的因素有对断接操作的支持、对跨区长事务的支持、对位置相关查询的支持、对查询优化的特殊考虑以及对提高有限资源的利用率和对系统效率的考虑等。

4．嵌入式移动数据库管理系统的应用

嵌入式数据库管理系统主要用于以移动信息设备为终端、并需要定期汇总的金融、零售、医疗、公安、保险、工业制造、仓储以及电信等多个行业和领域。

5．嵌入式数据库管理系统案例

SQL Anywhere Studio 是 Sybase 公司开发的一个嵌入式数据库系统，主要用于笔记本式计算机、手持设备和智能电器等领域。

Adaptive Server Anywhere 嵌入式数据库管理系统具有支持多种操作系统、支持 Java、支持 Internet、支持多种应用程序接口、易于管理及系统规模配置灵活等主要特性。

19.1.6　嵌入式网络及其他

嵌入式网络是用于连接各种嵌入式系统，使之可以互相传递信息、共享资源的网络系统。嵌入式系统在不同的场合采用不同的连接技术，如在家庭居室采用家庭信息网，在工业自动化领域采用现场总线，在移动信息设备等嵌入式系统则采用移动通信网。此外，还有一些专用连接技术用于连接嵌入式系统。

1．现场总线网

现场总线（Field Bus）是将数字传感器、变换机、工业仪表及控制执行机构等现场设备与工业过程控制单元、现场操作站等相互连接而成的网络。它具有全数字化、分散、双向传输和多分支的特点，是工业控制网络向现场级发展的产物。

嵌入式现场控制系统将专用微处理器置入传统的测量控制仪表，使其具备数字计算和数字通信能力。

现场总线主要有总线型与星型两种拓扑结构。现场总线控制系统通常由以下部分组成：现场总线仪表、控制器、现场总线线路、监控、组态计算机，并通过现场总线网卡、

通信协议软件连接到网上。

现场总线控制系统的优点如下。

（1）全数字化。

（2）全分布。

（3）双向传输。

（4）自诊断。

（5）节省布线及控制室空间。

（6）多功能。

（7）开放性。

（8）互操作性。

（9）智能化与自治性。

具有代表性的实例有德国 BOSCH 公司的 CAN（Control Area Network），Echelon 公司的 LONGWORKS。

2．家庭信息网

家庭信息网是把家庭范围内的个人计算机，家用电器，水、电、气仪表，照明设备和网络设备及安全设备等连接在一起的局域网。其主要功能是集中控制上述设备并将其接入 Internet，以共享网络资源和服务。

家庭信息网需要解决的两个基本问题如下。

（1）如何将家用电器，水、电、气仪表，照明设备等互相连接起来。

（2）如何实现这些连在一起的设备之间的互操作。

家庭信息网的拓扑结构有总线型和星型等。

目前，家庭信息网的传输技术有两类：一类是有线连接技术，包括以太网、电话线、电力线、IEEE1394 以及 USB 等。另一类是无线连接技术，包括蓝牙（B1ueTooth）、红外线（InfraRed）、无线 USB 以及 802.11 x 等相关无线标准。

3．无线数据通信网

无线数据通信网是一种通过无线电波传送数据的网络系统。通过无线数据通信网，智能手机、PDA 及笔记本式计算机可以互相传递数据信息，并接入因特网。

无线数据通信网分为短程无线网和无线因特网。短程无线网主要包括 802.11、蓝牙、IrDA 及 HomeRF 等。无线因特网或移动因特网主要采用两种无线连接技术：一种是移动无线接入技术，如 GSM（Global System for Mobile）、GPRS（General Packet Radio Service）和 CDPD（Cellular Digital Packet Data）等。另一种是固定无线接入技术，包括微波、扩频通信、卫星及无线光传输等。

无线局域网（Wireless LocalArea Network，WLAN）是计算机网络与无线通信技术相结合的产物。无线局域网的传输媒体是红外线（IR）或者无线电波（RF），目前无线电波的使用更广泛一些。

目前常见的无线网络标准以 IEEE 802.11x 系列为主。

4．嵌入式因特网

随着 Internet 和嵌入式技术的飞速发展，越来越多的信息电器都要求与 Internet 连接，来共享 Internet 所提供的方便、快捷、无处不在的信息资源和服务，即嵌入式 Internet 技术。嵌入式 Internet 技术在智能交通、家政系统、家庭自动化、工业自动化、POS 及电子商务等领域具有广阔的应用前景。

嵌入式因特网有直接接入因特网和通过网关接入因特网两种接入方法。

5．嵌入式系统的其他支撑软件

嵌入式系统的支撑软件通常包括窗口系统、网络系统、数据库管理系统及 Java 虚拟机等几个部分。

6．嵌入式窗口系统

嵌入式窗口系统是用于控制嵌入式系统中的位映像显示设备与输入设备的软件系统，管理屏幕、窗口、字体、光标、图形图像以及输入设备等资源。

图形用户界面系统是指计算机系统以图形方式向用户提供的人机交互的操作环境，如图 19-7 所示。

7．嵌入式窗口系统实例分析

嵌入式系统往往是一种定制的设备，它们对图形用户界面的需求也各不相同，因此很多嵌入式系统需要自己特定的嵌入式图形用户界面。常用的嵌入式图形用户界面有 MiniGUI、Microwindows、OpenGUI 和 Qt/Embedded。

8．嵌入式系统的 Java 虚拟机

Java 最初是由 Sun 公司开发的编程语言，可以在网络环境下为不同类型的计算机和操作系统开发软件。目前，在智能手机、机顶盒等嵌入式系统中得到了广泛的应用。

运行 J2ME（Java 2 Platform Micro-Edition）微型版的嵌入式设备主要有两大类：受限连接设备（Connected Limited Device）和连接设备（Connected Device）。如图 19-8 所示，J2ME 的结构分为 4 个层次：框架、配置、Java 虚拟机及嵌入式操作系统，J2ME 的核心是 Java 虚拟机。



图 19-7　图形用户界面系统的层次模型

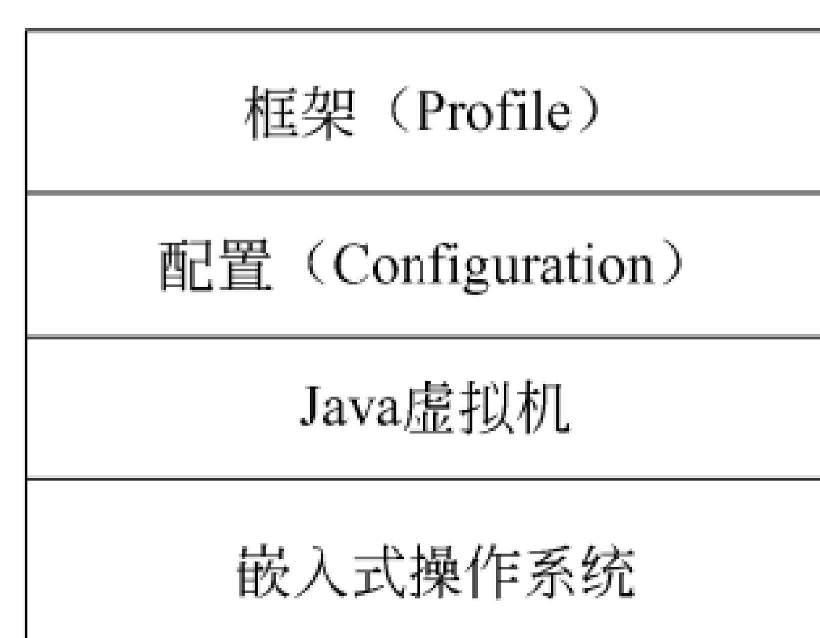


图 19-8　J2ME 的层次结构

KVM（K Virtual Machine）和 CVM（C Virtual Machine）都是 JVM 的子集，均可被看做是一种 Java 虚拟机，它们是 J2SE JVM 的压缩版。框架、配置、虚拟机与嵌入式操作系统之间的关系如图 19-9 所示。

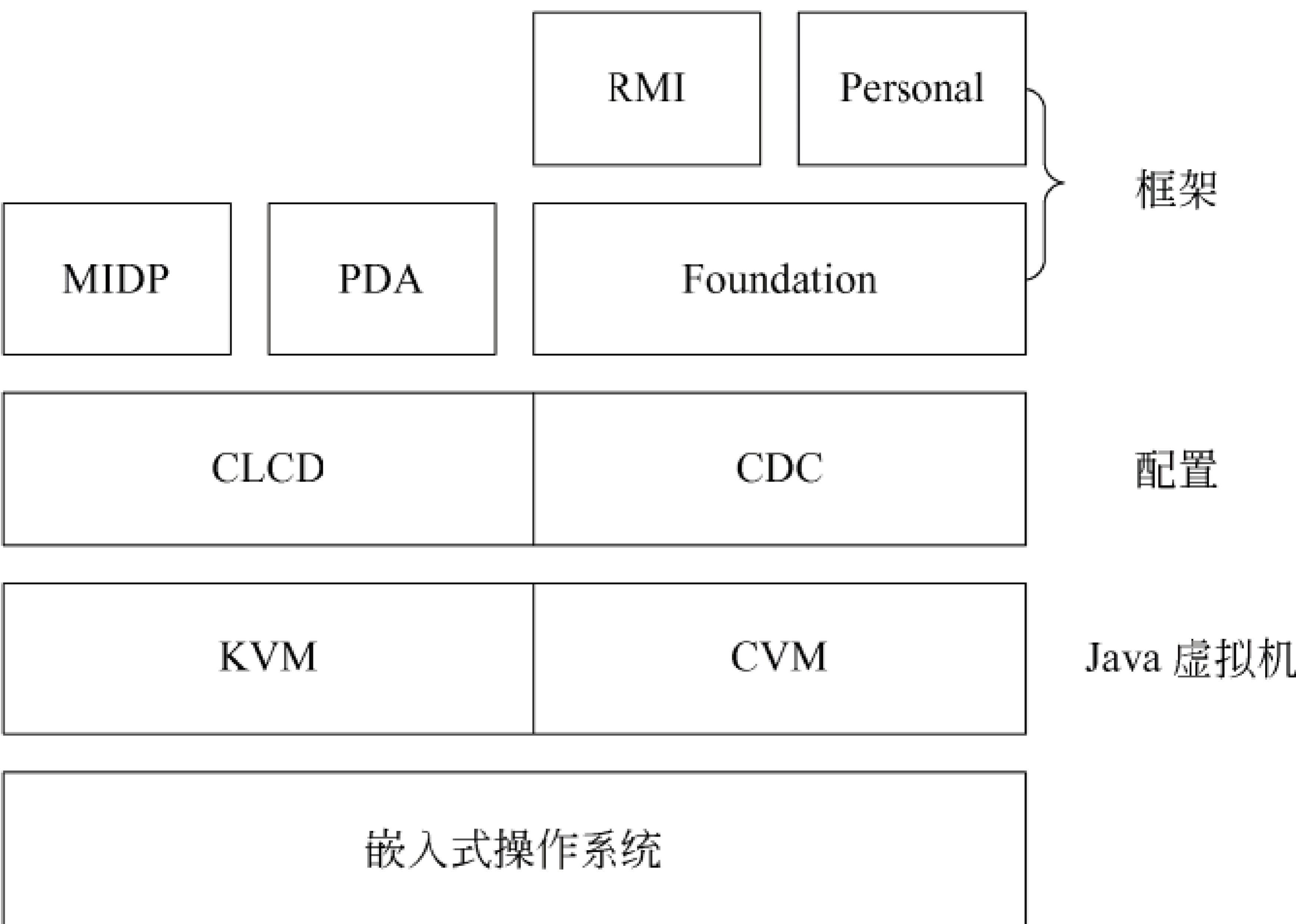


图 19-9 框架、配置、虚拟机与嵌入式操作系统之间的关系

目前，在 J2ME 中，主要有 MIDP、Personal、PDA、Foundation 和 RMI 等框架。

移动信息设备框架（Mobile Information Device Profile，MIDP）是一个 Java API 集合，它处理诸如用户界面、持久存储和联网等问题。

19.2 嵌入式系统的设计

19.2.1 嵌入式系统分析与设计

1．嵌入式系统设计的核心技术

嵌入式系统的核心技术有三种：处理器技术、IC 技术和设计/验证技术。

1）处理器技术

处理器技术与实现系统功能的计算引擎结构有关，不可编程的数字系统也可以视为处理器，这些处理器的差别在于其面向特定功能的专用化程度，导致其设计指标与其他处理器不同。

（1）通用处理器。

这类处理器可用于不同类型的应用，一个重要的特征就是存储程序，由于设计者不知道处理器将会运行何种运算，所以无法用数字电路建立程序。另一个特征就是通用的数据路径，为了处理各类不同的计算，数据路径是通用的，其数据路径一般有大量的寄

存器以及一个或多个通用的算术逻辑单元。设计者只需要对处理器的存储器编程来执行所需的功能，即设计相关的软件。

在嵌入式系统中使用通用处理器具有设计指标上的一些优势：上市时间快和成本较低；设计者只需编写程序，而不需要做任何数字电路设计；灵活性高，功能的改变通过修改程序进行。与自行设计处理器相比，小批量时单位成本较低。

当然，这种方式也有一些设计指标上的缺陷，数量大时的单位成本相对较高。因为数量大时，自行设计的成本分摊下来，可降低单位成本。同时，对于某些应用，性能可能很差。由于包含了非必要的处理器硬件，系统的体积和功耗可能变大。

（2）单用途处理器。

单用途处理器是设计用于执行特定程序的数字电路，也指协处理器、加速器和外设等。如 JPEG 编码解码器执行单一程序，压缩或解压视频信息。嵌入式系统设计者可通过设计特定的数字电路来建立单用途的处理器，也可以采用预先设计好的商品化的单用途处理器。

在嵌入式系统中使用单用途处理器，在指标上有一些优缺点。这些优缺点与通用处理器基本相反，性能可能更好，体积与功率可能较小，数量大时的单位成本可能较低，而设计时间与 NRE 成本可能较高，灵活性较差，数量小时的单位成本较高，对某些应用性能不如通用处理器。

（3）专用处理器。

专用指令集处理器（ASIP）是一个可编程处理器，针对某一特定类型的应用进行最优化。这类特定应用具有相同的特征，如嵌入式控制、数字信号处理等。在嵌入式系统中使用 ASIP，可以在保证良好的性能、功率和大小的情况下，提供更大的灵活性，但这类处理器仍需要昂贵的 NRE 成本建立处理器本身和编译器。单片机和数字信号处理器是两类应用广泛的 ASIP，数字信号处理器是一种针对数字信号进行常见运算的微处理器，而单片机是一种针对嵌入式控制应用进行最佳化的微处理器。通常控制应用中的常见外设，如串行通信外设、定时器、计数器、脉宽调制器及数/模转换器等都集成到了微处理器芯片上，从而使得产品的体积更小、成本更低。

2）IC 技术

（1）全定制/VLSI。

在全定制 IC 技术中，需要根据特定的嵌入式系统的数字实现来优化各层设计人员从晶体管的版图尺寸、位置、连线开始设计，以达到芯片面积利用率高、速度快、功耗低的最优化性能。利用掩膜在制造厂生产实际芯片，全定制的 IC 设计也常称为大规模集成电路设计（VLSI），具有很高的成本，很长的制造时间，适用于大量或对性能要求严格的应用。

（2）半定制 ASIC。

半定制 ASIC 是一种约束型设计方法，包括门阵列设计法和标准单元设计法。它是

在芯片上制作一些具有通用性的单元元件和元件组的半成品硬件，设计者仅需要考虑电路的逻辑功能和各功能模块之间的合理连接即可。这种设计方法灵活方便、性价比高，缩短了设计周期，提高了成品率。

（3）可编程 ASIC。

可编程器件中所有各层都已经存在，设计完成后，在实验室里即可烧制出设计的芯片，不需要 IC 厂家参与，开发周期显著缩短。可编程 ASIC 具有较低的成本，单位成本较高，功耗较大，速度较慢。

3）设计/验证技术

嵌入式系统的设计技术主要包括硬件设计技术和软件设计技术两大类。其中，硬件设计领域的技术主要包括芯片级设计技术和电路板级设计技术两个方面。

芯片级设计技术的核心是编译/综合、库/IP、测试/验证。编译/综合技术使设计者用抽象的方式描述所需的功能，并自动分析和插入实现细节。库/IP 技术将预先设计好的低抽象级实现用于高级。测试/验证技术确保每级功能正确，减少各级之间反复设计的成本。

软件设计技术的核心是软件语言。软件语言经历了从低级语言（机器语言、汇编语言）到高级语言（如结构化设计语言、面向对象设计语言）的发展历程，推动其发展的是汇编技术、分析技术、编译/解释技术等诸多相关技术。

早期，随着通用处理器概念的逐渐形成，软件技术迅速发展，软件的复杂度也开始增加，软件设计和硬件设计的技术和领域完全分开。设计技术和工具在这两个领域同步得到发展，也使得行为描述可以在越来越抽象的级别上进行，以适应设计复杂度不断增长的需要。采用 UML 等建模，进行分析与设计已成为统一共识。

2．系统的设计流程

嵌入式系统软件的开发过程可以分为项目计划、可行性分析、需求分析、概要设计、详细设计、程序建立、下载、调试、固化、测试及运行等几个阶段，这些阶段的次序和关系如图 19-10 所示。

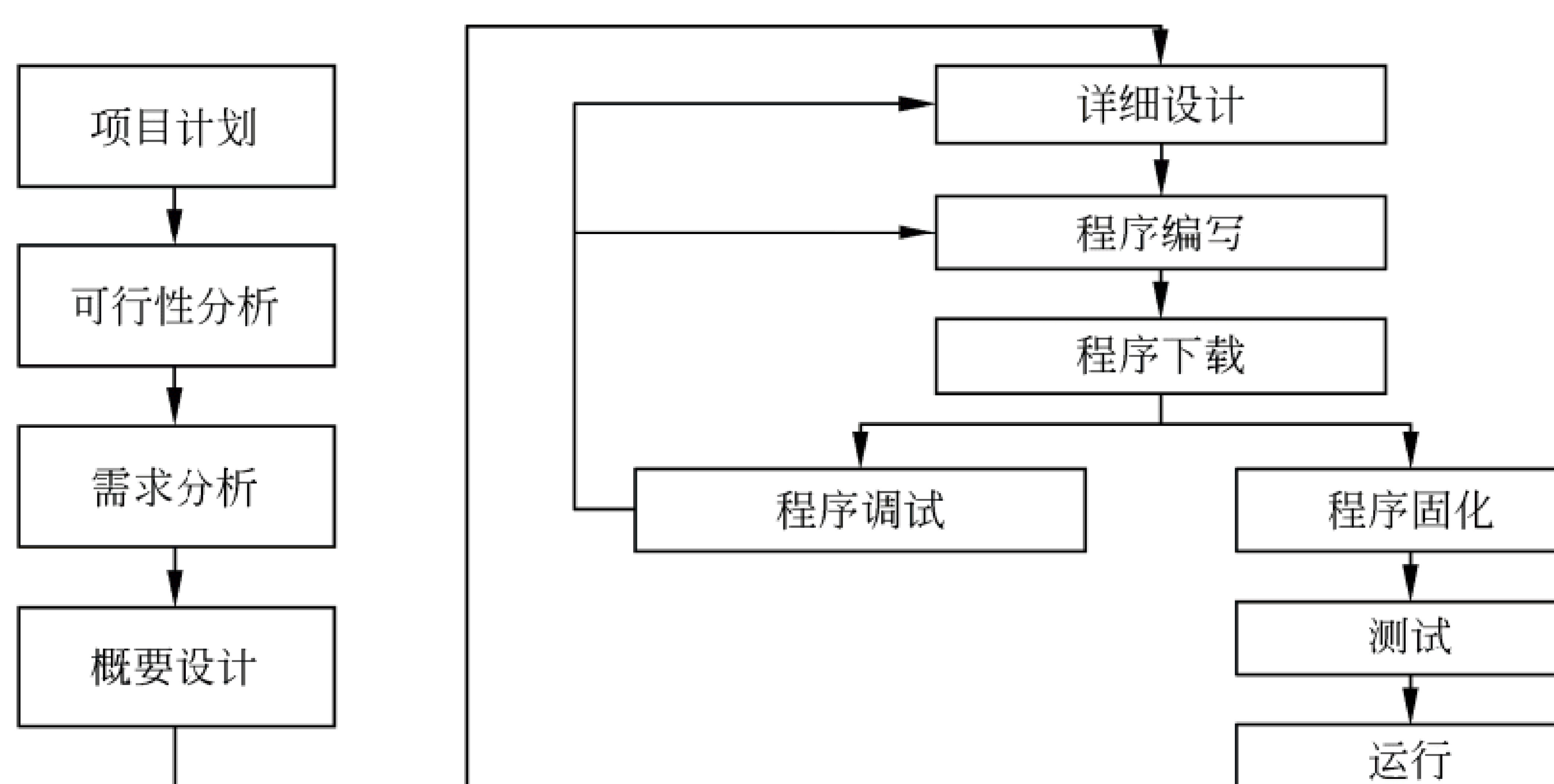


图 19-10　嵌入式软件开发的一般过程

（1）评估用户的需求采用计算机的必要性。从经济效益和社会效益这两个侧面来考虑。

（2）用户需求调查与分析，并提炼出规格说明。

（3）选择处理器。在用户需求调查清楚的基础上，再仔细进行分析。理解满足用户需求所需什么样的速度、什么样的精度、什么样规模的嵌入式系统可以实现用户的需求。在此基础上，首先确定作为系统的核心部件的处理器。选择合适的处理器对实现用户需求、提高系统性能、降低系统成本以及缩短开发周期都是十分重要的。选择处理器可从字长、速度、中断能力、环境的适应能力、硬件和软件的支持能力、开发和调试手段几个方面来考虑。

（4）制订系统方案。在系统的核心部件处理器确定后，便可以根据需求来制订系统的总体方案。这包括硬件系统方案和软件系统方案两个方面，并对总体方案进行评审。

（5）软、硬件分别设计。

（6）实验室联调。

（7）现场调试和试运行。

（8）鉴定或验收。

1）需求分析阶段

当确定要为用户设计开发嵌入式计算机系统后，接下来重要的一步就是对用户的需求进行认真仔细的调查和分析。这一步极为重要，因为此后系统设计的所有工作都是以用户的需求为依据的。用户的需求没有做好，则设计一定是失败的；用户没有要求的功能在设计中做了，有可能是锦上添花，也可能是画蛇添足。因此，最重要的在于满足用户的需求。

用户的需求调查一定要仔细进行，全面详细地了解要求，仔细地倾听用户的解释，经分析以文字的形式写出来并形成文档。而且使设计者的理解与用户的解释完全一致，不能存在二义性。

在需求调查时，除了仔细了解用户的需求外，还要对用户使用嵌入式系统的环境进行调查。这些内容也许用户没有意识到它们的重要性，但系统设计者必须予以高度重视。

一般都将形成文档的用户需求报告作为系统开发研制合同的附件，因为它既是开发的依据，又可在出现争议时备查。

2）规格说明

对用户需求进行提炼便可得到系统的规格说明。规格说明里包含了进行系统体系结构设计所需要的足够信息。将客户的描述转化为系统设计者的描述的结构化方法就是从客户的需求中获取一组一致性的需求，然后从中整理出正式的规格说明。

规格说明起到客户和生产者之间合同的作用，所以规格说明必须小心编写，以便精确地反映客户的需求并且作为设计时必须明确遵循的要求。

规格说明还应该足够明晰，以便别人可以验证它是否符合系统需求并且是否能完全

满足客户的期望，它亦不能有歧义。

3）设计阶段

稍具规模的嵌入式系统设计过程都是系统工程，需多个技术人员齐心协力共同以最快的速度加以实现。单靠一个人，即使有能力解决所有问题，但一个系统设计做出来可能就需好多年。当几年后系统完成时也就成为落后和无用的东西了，因为这个领域的技术发展非常快。

在硬件好分割的地方将硬件分割成若干模块。在软件好分割的地方、界面比较简单的地方将软件分割成若干模块，然后将硬件模块和软件模块分别交给不同的技术人员同时进行设计。

在设计过程中，规定设计进度，限定时间将各自的模块设计并调试出来。在此过程中，对出现的问题要进行协调。若有方案上的变更，要以文字的形式通知设计人员。各设计人员在设计和调试自己的模块过程中要做必要的记录。

4）系统集成与测试阶段

当软、硬件各模块都设计调试完成后，便可在实验室进行联调。

将硬件模块逐块加到硬件系统上，逐块模板进行调试。直到所有硬件模块都调试出来，证明它们可以正常工作。

同时，对各自设计的软件模块逐块进行连接并调试，证明软件系统可以工作。

将软、硬件结合在一起，对整个系统进行调试，并在实验室里进行模拟试运行。在实验室模拟试运行中，如果需要，可在实验室里产生模拟信号。例如，称重传感提供 0～20mV 的称重信号，也许实验室里没有称重传感器，但实验室里一定可以产生 0～20mV 的信号代替传感器的输出。同样，实验室里可能没有需要 0～5mV 的执行机构，但用万用表便可测量出系统硬件是否能够输出执行机构所需要的 0～5mV 的信号。

经过不断的模拟试运行，仔细观察并分析出现的现象和状态。判断系统工作是否正常，决不要放过任何的异常，对出现的问题随时加以解决。

由于用户需求的复杂性、多样性，使得系统的硬件和软件变得复杂得多。因此，在实验室里进行模拟试运行时，一定要想办法使用户程序的每一条路径都能走到，即测试用户程序的各种可能。同时，要使系统连续运行较长的时间，以便发现可能隐藏的软、硬件故障。

5）现场调试和试运行

在实验室模拟试运行确认系统工作是正确的情况下，便可将系统运往用户现场，在现场进行安装和调试。这时，系统所连接的专用外设都是系统真正要使用的，在调试时也应逐个进行外设的连接，逐个进行调试。

在调试通过后，即可使系统开始试运行。在试运行的过程中，一定要密切注视系统运行的状态，不要放过任何的异常情况；对系统运行的状态和数据进行详细记录；如果出现问题，应立即予以解决；要使系统无故障地、正常运行半年或更长的时间。

如果可能，最好多做几套样机，并将它们安装在不同的用户现场上，进行半年或更长时间的试运行，以便确定系统的工作性能。

19.2.2　嵌入式软件设计模型

1．状态机模型

有限状态机（FSM）是一种描述系统状态及其状态转换的节点网，包括节点和边，节点表示状态，边表示状态之间的转换关系。边上面标注事件，表示状态转换对该事件敏感。在一个典型的有限状态机中，系统总是处于单个状态，事件通信可以是广播、同步和非阻塞方式。图 19-11 显示了一个有限状态机的示例。

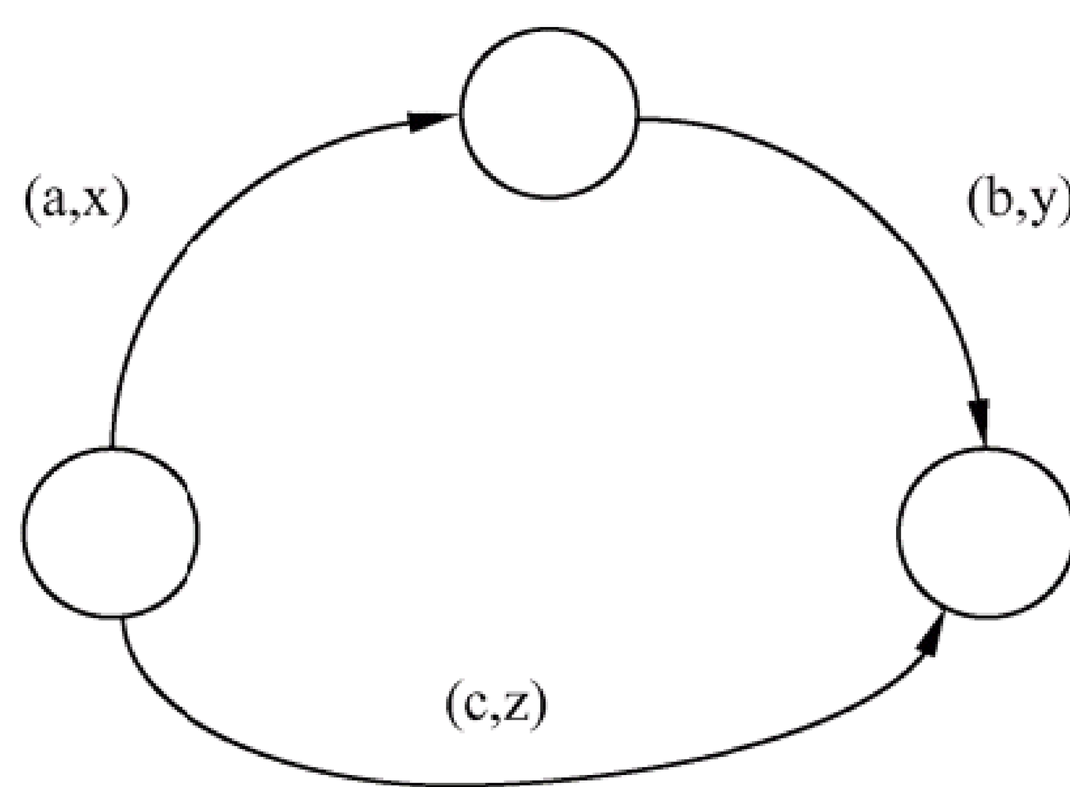


图 19-11　有限状态机示例

有限状态机适合于对控制领域的系统建模。但是，由于缺乏并发和层次化支持，创建系统的规模受限，无法对并发系统建模，这是由于有限状态机总是假设系统处于单个状态而造成的。

状态机被描述为：输入事件的集合；输出事件的集合；状态集合；把状态和输入事件映射到输出事件的集合；把状态和输入事件映射到状态的集合；对初始状态的描述；有限状态机是有限状态的机器。有限状态机可被用作提出和解决问题的开发工具，也可被用作为以后开发者描述解决方案的正式工具。有很多种显示状态机的方法，从简单的表格到图形示例。所谓状态机模型，是一个描述状态变迁的方法，它总是将一种状态向另一种状态的变迁视为由输入消息激励所产生的结果。这样，每当消息促使事务需要做出某种操作时，有限状态机的当前状态也随之改变，这种改变是根据预先制定好的规则来实现的。

状态机模型具有如下特点。

（1）输出由当前的输入和当前的状态决定。这是状态机模型和组合模型之间的最大区别。

（2）当前的状态由过去的输入决定。很显然，一个对象之所以拥有当前状态，是因为它在此前经历过初始化并响应过若干次外界的操作请求（即接受了外界的输入）。

（3）当前的状态是上一步输出的反映。被测对象处于当前状态的直接原因是对象对上一次的输入做了有效的回应，即产生了上一步的输出。

（4）输出受输入顺序的影响。因为对状态机模型而言，当前状态加上当前输入才可以得到预期的输出。如果把当前输入放在上一步输入之前，那时状态机既然不是处于当前状态之下，就无法得到预期的输出了。

2．数据流模型

数据流图允许系统作为操作网进行建模，特别适合于对实现进行分区的系统模型。

这些模型的长度能够描述系统控制和数据域，这使得它们能够适合于对异构系统的建模。

数据流模型主要包括布尔数据流、层次化的流图和 Petri 网。

1）布尔数据流

布尔数据流克服了同步数据流模型中数据依赖操作的缺陷，引入了开关结点，能够根据输入值选择输出，支持对复杂算法的建模和实现。图 19-12 显示了一个布尔数据流模型。

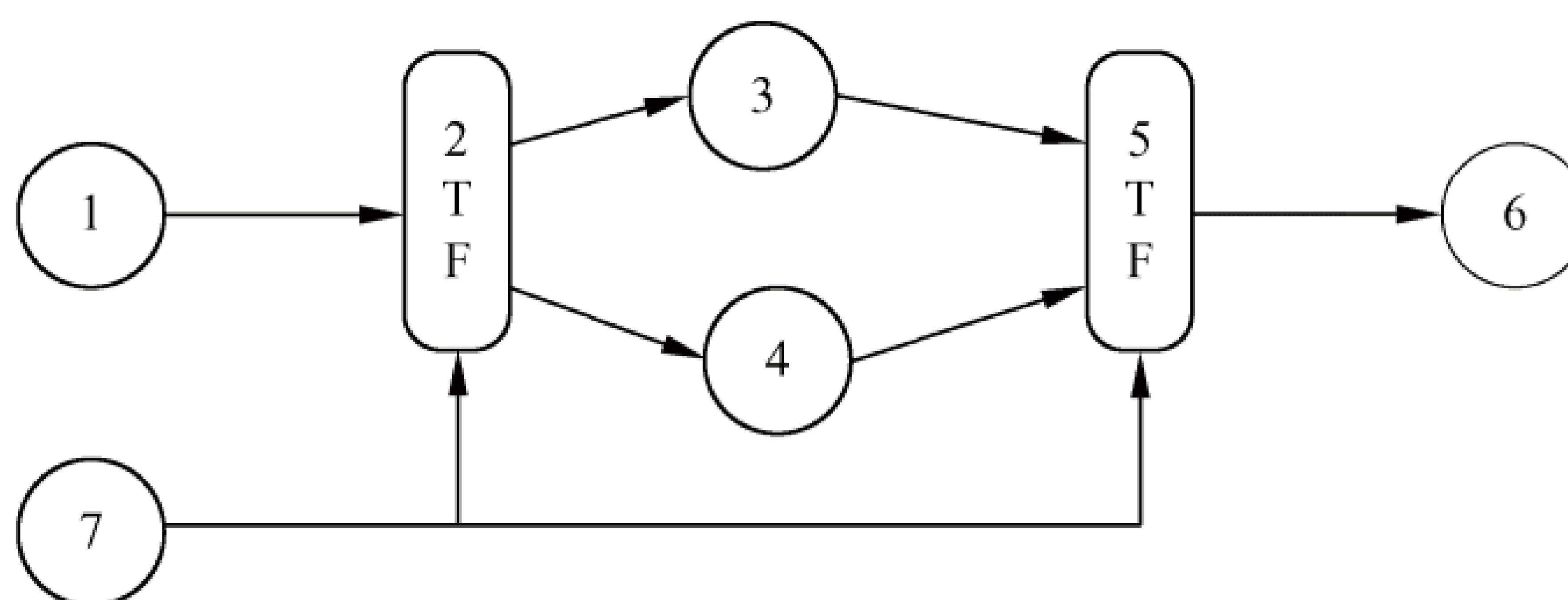


图 19-12　布尔数据流模型示例

2）层次化的流图

层次化的流图包括节点和边，节点表示操作，边表示依赖。操作可以是条件、计算、等待和循环操作。条件操作支持在图模型中使用数据依赖。通信可以是阻塞式（Wait）和非阻塞式（Receive）。层次化的流图用于在系统中执行分区、调度和综合。

3）Petri 网

Petri 网是一种流程建模机制，其基本形式是由 Petri 提出的，所以命名为 Petri 网。目前，Petri 网得到了广泛应用，特别是在离散事件仿真、实时调度、工业控制和工作流等方面。基本 Petri 网通常包括库所（Places）、操作（Transitions）和有向边（弧）。库所上的数据称为令牌（Tokens）。输入库所上的令牌可以触发迁移，消耗输入库所上的令牌，并为输出库所生成令牌。图 19-13 显示了一个 Petri 网的示例。

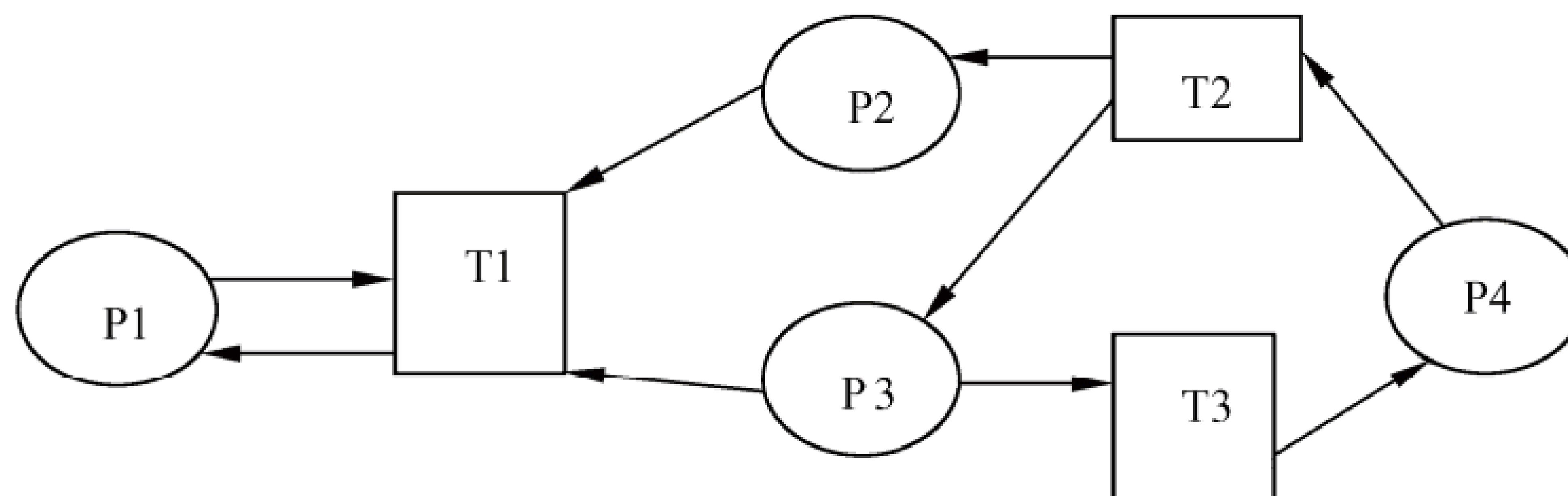


图 19-13　Petri 网示例

3．并发进程模型

并发进程模型包括 CSP 与 CCS 等。

1）CSP 模型

CSP（Communicating Sequential Processes）通信顺序进程是 C.A.R.Hoare 于 1978 年提出的一种并发、分布式程序设计语言模型。CSP 一经出现就被广泛地应用于计算机科学的诸多领域，如网络通信协议的形式化描述等。

CSP 将输入、输出操作列为程序语言的基本要素，而将实现顺序进程间通信的并行组合作为基本的程序控制结构。用这种语言设计的一个程序，就是一组进程，它们通过一个通信网络彼此通信。

CSP 模型的目的是描述一种在计算机应用的广泛领域中适用的最简单的数学理论，其主要贡献是把计算机所涉及的各种计算形式及其性质建立在一套严密的形式系统上。其新版本 TCSP 在并发和通信方法及语义的研究方面与 Milner 的 CCS 很类似，对“失效语义”给出了精确的数学描述，使 CSP 理论更加可靠完善。

2）CCS 模型

CCS（Calculus of Communicating System）通信演算系统是 R.Milner 于 1980 年发表的一个建立于极小原语集上的函数式程序设计语言模型，为通信的、非确定的并发系统提供了一个通用数学模型。

CCS 模型是在一种较弱条件下建立起来的并发进程模型，企图俘获并发性及通信的一般数学性质。其主要贡献是关于并发系统构成的等价性研究，其中有代表性的是建立在双模拟基础上的等价概念。

CCS 从简单的事实出发，以严谨、优美的数学形式，建立了并发系统行为的形式理论。

4．面向对象模型

面向对象模型的出发点，就是要把现实世界中物与物的关系怎样不变地用程序表示出来，并把现实世界的组织结构在计算机上再现。这种结构直观、易懂。现实世界中的万物都有自己的功能和任务，自己能干的事情自己完成，自己不能干的事情托他人代办，只要取得结果就可以了。因此，就要注意现实世界中各事物——即对象之间的联系，为每个对象分配任务，对象之间也相互传递任务，进行工作。这就是面向对象的方法。

面向对象的基本结构可用 6 个术语来描述，即对象、类、属性、消息、操作和关系。系统的结构中心是对象，每个对象中有属性和操作，属性和操作封装在一个盒子里。对象之间通过传送消息来协调工作。每个对象又进一步抽象为类，类是对象实体的模板。对象之间的相互地位用关系表示。关系大致分为三大类，即关联关系、Part-Of 关系和 Is-a 关系。关联关系表示对象之间对等访问或利用关系。对象之间存在关联，表示相连的对象之间可以传递消息。Part-Of 关系表示一方是他方对象的一部分的包含关系，是对象间的集约关系。Is-a 关系则是类间的包含关系。在 Is-a 关系中，子类既继承父类的性质，又有自己独有的性质。这里的性质是指属性、操作和关系。

面向对象的模型特征可以从以下几个方面体现。

1）抽象化

通过对象来抽象现实世界有两点好处：首先是能自然地表示现实世界，通过对现实

世界的模仿，就可以类推出需要的功能和操作；其次，是很容易明确分析焦点，只要把现实世界的一个重要侧面模型化，在以后的分析中必须注重的焦点就自然明确了。抽象化除把现实世界的事物抽象为对象外，还把性质相同的对象群进一步抽象为类。通过这种层次式的抽象，便可构造事物的体系，很自然地把握现实世界。

2）封装化

封装化可以把对象内部的数据和操作过程隐藏起来，可以控制模块间信息公开和隐藏的范围。对外，只让看到过程的规格说明，使得对象的规格说明和实现相分离。封装化的最大效果是把对象的提供者和对象的使用者分开，对象的使用者只知道在对象中定义的操作的规格说明，对象内部的数据结构和操作过程是不知道的。有些过程群是为了访问特定的数据结构而设置的，可以集中放在类中，也不会影响到对象使用者的信息，为需求的变更带来方便，同时还可以拒绝非法使用者的访问，达到保密的目的。

3）继承化

面向对象模型中，父类的概念可继承到子类，且子类还可以有自己的新性质，这就叫继承性。继承性可有效地重用资源、提高生产效率，具体体现在两点上：一是可以把类作为体系化的手段，即多个子类的共同性质抽象为一个父类。由此，父类容易明白子类之间性质的不同及分类观点，也便于子类追加自己的性质。二是可利用继承性来定义新类，只要定义出与现有类的不同点就可以了，有利于产品重用和更改管理。

4）状态

对象中所定义的过程能否使用，实际上是通过内部的属性值来决定的。根据属性值，可以把对象分成多个不同的状态。每种状态下，都标识出此刻可使用的过程。把状态的概念放入对象规格说明中，使得对象的提供者能把正确接收消息的过程作为状态转移图，提供给使用者。

由面向对象模型的分析可看出，面向对象方法论的特征是能在早期获取模型结构和现实世界结构的对应关系，所有的软件都可以看成是现实世界的模拟。

类—责任—协作者（Class-Responsibility-Collaborator，CRC）模型是面向对象建模的对象描述工具，用于标识类、指明类的责任（属性和操作）以及类之间的协作（多个类协同完成某些操作）。面向对象模型还包括对象信息、行为模型等，它们从不同侧面（静态、动态）描述系统。

19.2.3 嵌入式系统软件开发环境

1. 嵌入式系统开发概述

嵌入式系统的软件开发采用交叉平台开发方法（Cross Platform Development），即软件在一个通用的平台上开发，而在另一个嵌入式目标平台上运行。这个用于开发嵌入式软件的通用平台通常叫做宿主机系统，被开发的嵌入式系统称为目标机系统。而当软件执行环境和开发环境一致时的开发过程则称为本地开发（Native Development）。

2．调试方法

在嵌入式系统的实际开发实践中，经常采用的调试方法有直接测试法、调试监控法、在线仿真法、片上调试法及模拟器法等。

3．开发环境分类

嵌入式系统的开发环境可以分为如下几类。

（1）与嵌入式操作系统配套的开发环境。

（2）与处理器芯片配套的开发环境。

（3）与具体应用平台配套的开发环境。

（4）其他类。

4．开发环境举例

目前业界应用最广泛的集成开发环境为 Tornadoll，美国 WindRiver 公司的 VxWorks 操作系统是 Tornadoll 嵌入式开发环境的关键组成部分。

第 20 章　面向服务的架构

Massimo Pezzini, Gartner Group 说过，“当有一天，所有的应用都写成 Web 服务，集成也许可以变得更容易”

服务是一个由服务提供者提供的，用于满足使用者请求的业务单元。服务的提供者和使用者都是软件代理为了各自的利益而产生的角色。

在 SOA 中，服务的概念有了延伸，泛指系统对外提供的功能集。例如，在一个大型企业内部，可能存在进销存、人事档案和财务等多个系统，在实施 SOA 后，每个系统用于提供相应的服务，财务系统作为资金运作的重要环节，也向整个企业信息化系统提供财务处理的服务，那么财务系统的开放接口可以看成是一个服务。

20.1　SOA 的相关概念

20.1.1　SOA 的定义

面向服务的体系结构（Service-Oriented Architecture，SOA），从应用和原理的角度看，目前有两种业界公认的标准定义。

从应用的角度定义，可以认为 SOA 是一种应用框架，它着眼于日常的业务应用，并将它们划分为单独的业务功能和流程，即所谓的服务。SOA 使用户可以构建、部署和整合这些服务，且无需依赖应用程序及其运行平台，从而提高业务流程的灵活性。这种业务灵活性可使企业加快发展速度，降低总体拥有成本，改善对及时、准确信息的访问。SOA 有助于实现更多的资产重用、更轻松的管理和更快的开发与部署。

从软件的基本原理定义，可以认为 SOA 是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以一种统一和通用的方式进行交互。

作为软件架构师，后一种从软件原理方面的定义，对日常工作更具指导性。

20.1.2　业务流程与 BPEL

业务流程是指为了实现某种业务目的行为所进行的流程或一系列动作。在计算机领域，业务流程代表的是某一个问题在计算机系统内部得到解决的全部流程。

由于业务流程来源于现实世界，传统上是通过复杂的语言进行描述。在计算机业务

系统建模中，需要用到一种特定的、简洁的语言来专门描述计算机系统的业务流程，这便促使了 BPEL 的诞生。

BPEL（Business Process Execution Language For Web Services）翻译成中文的意思是面向 Web 服务的业务流程执行语言，也有的文献简写成 BPEL4WS，它是一种使用 Web 服务定义和执行业务流程的语言。使用 BPEL，用户可以通过组合、编排和协调 Web 服务自上而下地实现面向服务的体系结构。BPEL 提供了一种相对简单易懂的方法，可将多个 Web 服务组合到一个新的复合服务（称作业务流程）中。

BPEL 目前用于整合现有的 Web Services，将现有的 Web Services 按照要求的业务流程整理成为一个新的 Web Services，在这个基础上，形成一个从外界看来和单个 Service 一样的 Service。

20.2　SOA 的发展历史

20.2.1　SOA 的发展历史

SOA 的概念最初由 Gartner 公司提出，由于当时的技术水平和市场环境尚不具备真正实施 SOA 的条件，因此当时 SOA 并未引起人们的广泛关注，SOA 在当时沉寂了一段时间。伴随着因特网的浪潮，越来越多的企业将业务转移到因特网领域，带动了电子商务的蓬勃发展。为了能够将公司的业务打包成独立的、具有很强伸缩性的基于因特网的服务，人们提出了 Web 服务的概念，这可以说是 SOA 的起源。

Web 服务开始流行以后，因特网迅速出现了大量的基于不同平台和语言开发的 Web 服务组件。为了能够有效地对这些为数众多的组件进行管理，人们迫切需要找到一种新的面向服务的分布式 Web 计算架构，该架构要能够使这些由不同组织开发的 Web 服务相互学习和交互，保障安全以及兼顾复用性和可管理性。由此，人们重新找回面向服务的架构，并赋予其时代的特征。需求推动技术进步，正是这种强烈的市场需求，使得 SOA 再次成为人们关注的焦点。回顾 SOA 发展历程，我们把其大致分为了三个阶段，下面将分别介绍每个阶段的重要标准和规范。

SOA 的发展最初始于国外，其经历了如下三个阶段。

1．萌芽阶段

这一阶段以 XML 技术为标志，时间大致从 20 世纪 90 年代末到 21 世纪初。虽然这段时期很少提到 SOA，但 XML 的出现无疑为 SOA 的兴起奠定了稳固的基石。

XML 系 W3C 所创建，源自流行的标准通用标记语言（Standard Generalised Markup Language，SGML），它在 20 世纪 60 年代后期就已存在。这种广泛使用的元语言，允许组织定义文档的元数据，实现企业内部和企业之间的电子数据交换。由于 SGML 比较复杂，实施成本很高，因此很长时间里只用于大公司之间，限制了它的推广和普及。

通过 XML，开发人员摆脱了 HTML 语言的限制，可以将任何文档转换成 XML 格式，然后跨越因特网协议传输。借助 XML 转换语言（eXtensible Stylesheet Language Transformation，XSLT），接受方可以很容易地解析和抽取 XML 的数据。这使得企业既能够将数据以一种统一的格式描述和交换，同时又不必负担 SGML 那样高的成本。事实上，XML 实施成本几乎和 HTML 一样。

XML 是 SOA 的基石。XML 规定了服务之间以及服务内部数据交换的格式和结构。XSD Schemas 保障了消息数据的完整性和有效性，而 XSLT 使得不同的数据表达能通过 Schema 映射而互相通信。

2．标准化阶段

2000 年以后，人们普遍认识到基于公共——专有因特网之上的电子商务具有极大的发展潜力，因此需要创建一套全新的基于因特网的开放通信框架，以满足企业对电子商务中各分立系统之间通信的要求。于是，人们提出了 Web 服务的概念，希望通过将企业对外服务封装为基于统一标准的 Web 服务，实现异构系统之间的简单交互。这一时期，出现了三个著名的 Web 服务标准和规范：简单对象访问协议（Simple Object Access Protocal，SOAP）、Web 服务描述语言（Web Services Description Language，WSDL）及通用服务发现和集成协议（Universal Discovery Description and Integration，UDDI）。

这三个标准可谓 Web 服务三剑客，极大地推动了 Web 服务的普及和发展。短短几年之间，因特网上出现了大量的 Web 服务，越来越多的网站和公司将其对外服务或业务接口封装成 Web 服务，有力地推动了电子商务和因特网的发展。Web 服务也是因特网 Web 2.0 时代的一项重要特征。

3．成熟应用阶段

从 2005 年开始，SOA 推广和普及工作开始加速。不仅专家学者，几乎所有关心软件行业发展的人士都开始把目光投向 SOA。一时间，SOA 频频出现在各种技术媒体、新产品发布会和技术交流会上。

各大厂商也逐渐放弃成见，通过建立厂商间的协作组织共同努力制定中立的 SOA 标准。这一努力最重要的成果体现在三个重量级规范上：SCA/SDO/WS-Policy。SCA 和 SDO 构成了 SOA 编程模型的基础，而 WS-Policy 建立了 SOA 组件之间安全交互的规范。这三个规范的发布，标志着 SOA 进入了实施阶段。

从整体架构角度看，人们已经把关注点从简单的 Web 服务拓展到面向服务体系架构的各个方面，包括安全、业务流程和事务处理等。

20.2.2 国内 SOA 的发展现状与国外对比

在 SOA 概念深入普及的同时，国内也兴起了对 SOA 的研究和初步实践。2007 年，IDC 发布了《SOA 中国路线图》。有观点认为，这是“国际研究机构首次基于中国 IT 背景，针对中国企业实施 SOA 路线所做的特定解读”，这也是目前关于 SOA 这一新兴技

术在中国实施的第一份比较权威的报告。

报告中，重点指出了中国和美国在 SOA 领域的差距。

在美国，过去的半个多世纪，美国从主机时代、PC 时代，到了现在的网络时代，积累了大量的应用系统。美国实现 SOA 架构的关键任务是：对已有系统中的功能进行提取和包装，形成标准的“服务”。

在中国，过去近 30 年的 IT 建设多为生产型系统，服务型系统普遍未开始建设，大量“服务”需要全新构造才是中国 SOA 的主要任务。

这份报告的可取之处如下。

（1）指出了中美 IT 系统所面临的根本性问题不同。现阶段，国内主要是以“构建支撑某一业务的应用系统”为主，其中伴随着一部分企业内部应用系统之间的整合。如果用前面的“三个阶段”来做以下匹配，可能更多还处于第一阶段，即使第二阶段的应用也处于起步状态。

（2）为国内大型集团化企业指明了如何解决系统集成和系统构建的融合性问题，基于 SOA 方式下的解决方案。

关于国内实施 SOA 的现状，报告用表 20-1 进行了阐述。

表 20-1　国内商用领域和政府领域的 SOA 应用

IT 建设领先领域 （电信、金融）	服务型系统还没开始大规模构造领域 （政府、电力、国防）
1．采用对遗留系统进行切割和封装的方式，或整个系统包装成一个服务。 2．未来的新建系统用粒度更小、组合更容易、架构更灵活的面向构件技术构造。 3．用 ESB 实现新旧“服务”的注册与管理	1．首先需要统一标准（SCA/SDO）。 2．用符合 SOA 标准的方法（面向构件）构造粒度更小、组合更容易、架构更灵活的“服务”。 3．SOA 的流程管理。 4．SOA 的软件治理。 5．多“服务”用 ESB 实现集成

20.3　SOA 的参考架构

IBM 的 Websphere 业务集成参考架构（如图 20-1 所示，以下称参考架构）是典型的以服务为中心的企业集成架构，接下来的讨论都将以此参考架构为背景进行。

以服务为中心的企业集成采用“关注点分离（Separation of Concern）”的方法规划企业集成中的各种架构元素，同时从服务视角规划每种架构元素提供的服务，以及服务如何被组合在一起完成某种类型的集成。这里架构元素提供的服务既包括狭义的服务（WSDL 描述），也包括广义的服务（某种能力）。从服务为中心的视角来看，企业集成的架构按图 20-1 所示的方式划分为 6 大类。

（1）业务逻辑服务（Business Logic Service）：包括用于实现业务逻辑的服务和执行

业务逻辑的能力，其中包括业务应用服务（Business Application Service）、业务伙伴服务（Partner Service）以及应用和信息资产（Application and Information asset）。

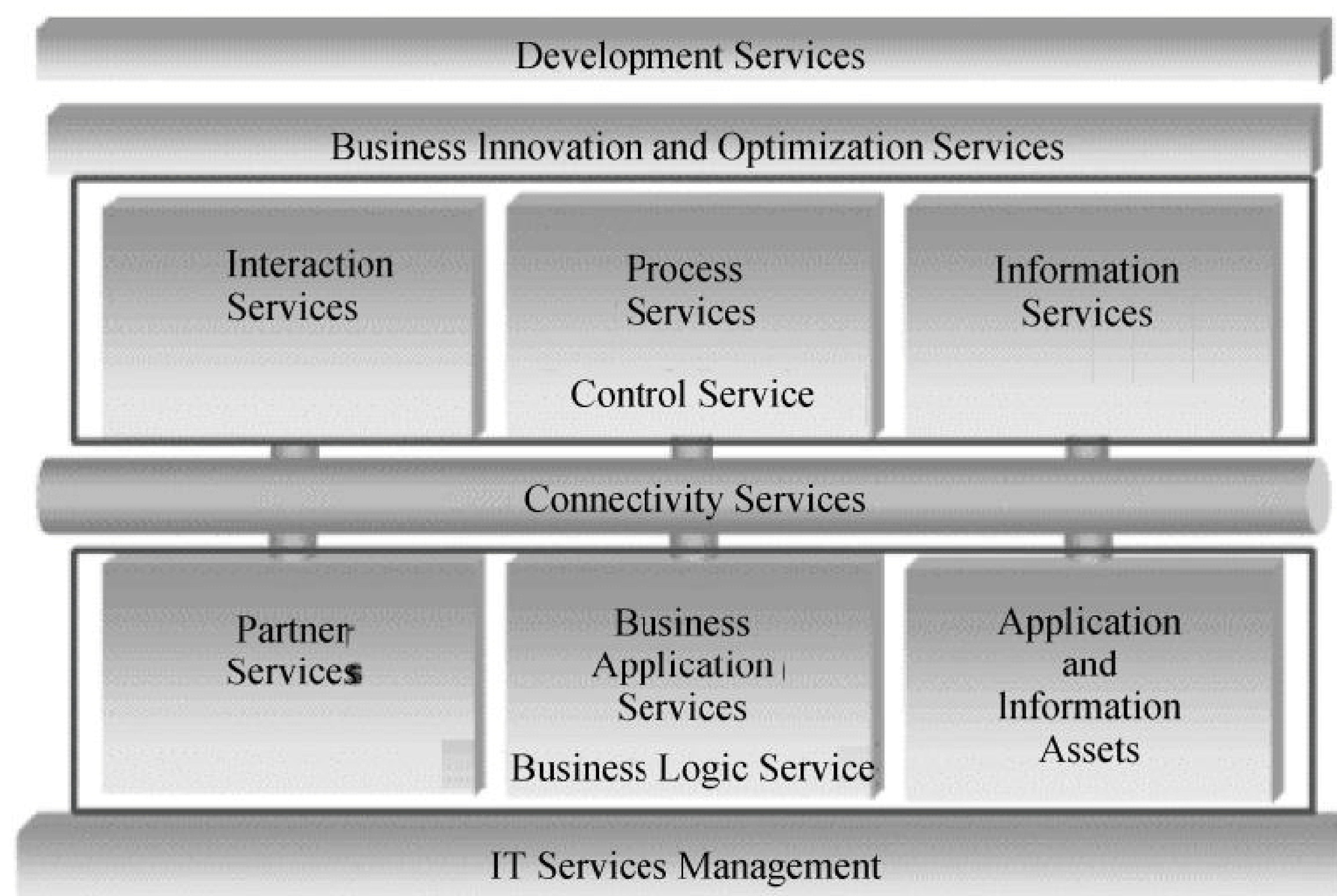


图 20-1　IBM WebSphere 业务集成参考架构

（2）控制服务（Control Service）：包括实现人（people）、流程（process）和信息（infor-mation）集成的服务，以及执行这些集成逻辑的能力。

（3）连接服务（Connectivity Service）：通过提供企业服务总线提供分布在各种架构元素中服务间的连接性。

（4）业务创新和优化服务（Business Innovation and Optimization Service）：用于监控业务系统运行时服务的业务性能，并通过及时了解到的业务性能和变化，采取措施适应变化的市场。

（5）开发服务（Development Service）：贯彻整个软件开发生命周期的开发平台，从需求分析，到建模、设计、开发、测试和维护等全面的工具支持。

（6）IT 服务管理（IT Service Management）：支持业务系统运行的各种基础设施管理能力或服务，如安全服务、目录服务、系统管理和资源虚拟化。

1．连接服务——企业服务总线

企业服务总线（Enterprise Service Bus，ESB）是过去消息中间件的发展，采用了“总线”这样一种模式来管理和简化应用之间的集成拓扑结构，以广为接受的开放标准为基础来支持应用之间在消息、事件和服务的级别上动态地互联互通。

ESB 的基本特征和能力包括：描述服务的元数据和服务注册管理；在服务请求者和提供者之间传递数据，以及对这些数据进行转换的能力，并支持由实践中总结出来的一些模式如同步模式、异步模式等；发现、路由、匹配和选择的能力，以支持服务之间的

动态交互，解耦服务请求者和服务提供者。高级一些的能力，包括对安全的支持、服务质量保证、可管理性和负载平衡等。

ESB 所提供的基于标准的连接服务，将应用中实现的功能或者数据资源转化为服务请求者能以标准的方式来访问的服务；当请求者来请求一个服务时，ESB 中这种中介转化过程可能简单到什么也没有，也可能需要很复杂的中介服务支持，包括动态地查找、选择一个服务，消息的传递、路由和转换、协议的转换。这种中介过程，是 ESB 借助于服务注册管理以及问题域相关的知识（如业务方面的一些规则等）自动进行的，不需要服务请求者和提供者介入，从而实现了解耦服务请求者和提供者的技术基础，使得服务请求者不需要关心服务提供者的位置和具体实现技术，双方在保持接口不变的情况下，各自可以独立地演变。

所以，ESB 采用总线结构模式简化了应用之间的集成拓扑，通过源自实践的模式，提供了基于标准的通用连接服务，使得服务请求者和服务提供者之间可以以松散耦合、动态的方式交互，从而在不同层次上使得 SOI 解决方案是一个松散耦合、灵活的架构。

一个典型的企业服务总线如图 20-2 所示。需要注意的是，ESB 是一种架构模式，不能简单地等同于特定的技术或者产品，但实现 ESB 确实需要各种产品在运行时和工具方面的支持。IBM 有很好的产品支持，运行时支持包括 WebSphere ESB 和 WebSphere Message Broker；而工具方面 IBM 则有 WebSphere Integration Developer，支持用户以图形界面的方式来完成相关的开发任务，如发布服务，使用各种模式、转换消息和定义路由等。

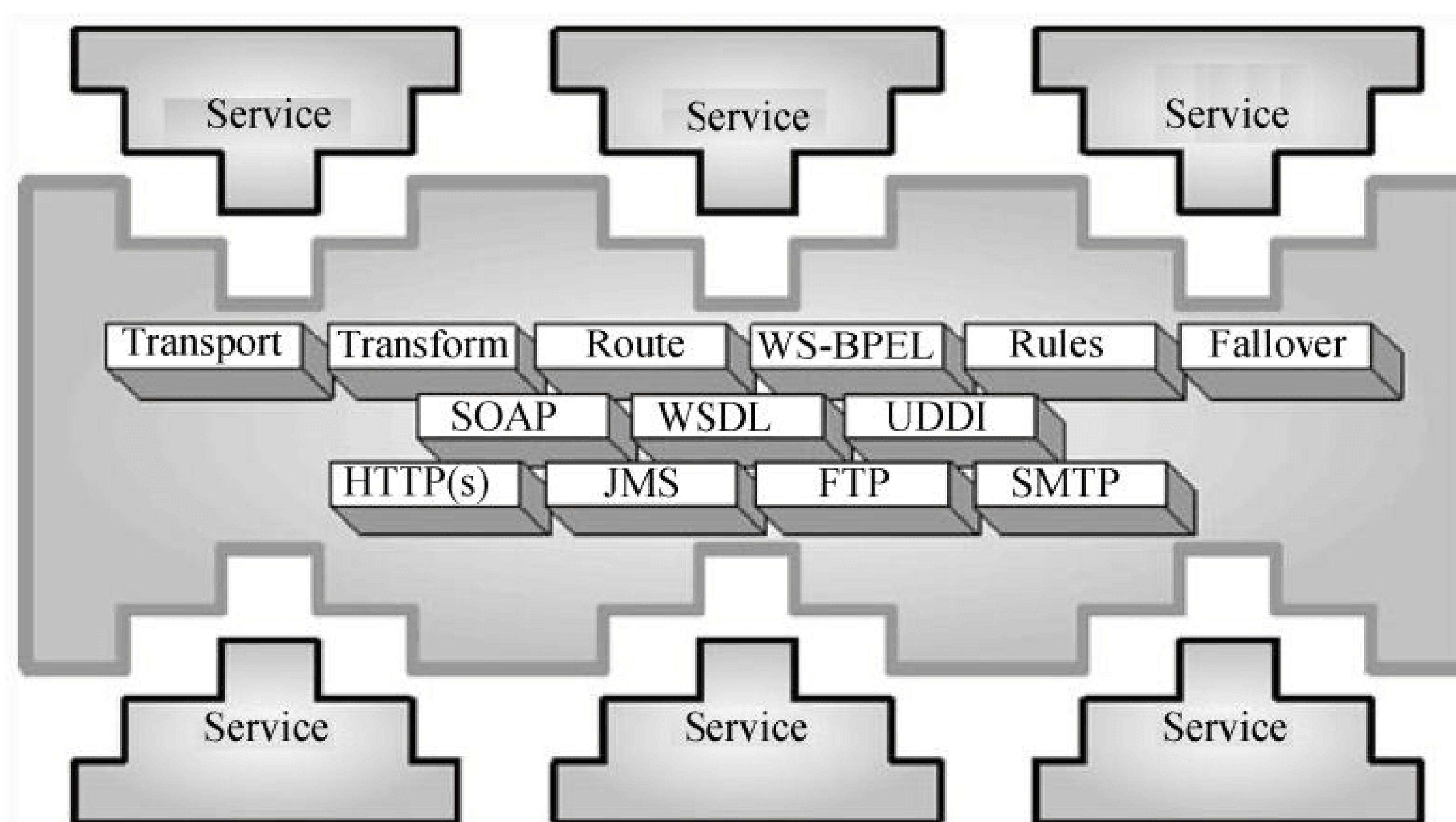


图 20-2　一个典型的企业服务总线

2．业务逻辑服务

1）整合已有应用——应用和信息访问服务

已有应用和信息是实现业务逻辑和业务数据的重要资产。通过集成已有的应用和信

息将可以在已有企业系统上实现更多增值服务，所以集成已有应用和信息是企业集成中重要的一环。

以服务为中心的企业集成通过应用和信息访问服务（Application and Information Access Service）来实现对已有应用和信息的集成。它通过各种适配器技术将已有系统中的业务逻辑和业务数据包装成企业服务总线支持的协议和数据格式。通过企业服务总线，这些被包装起来的业务逻辑和数据就可以方便地参与上层的业务流程，从而已有应用系统的能力可以得以继续发挥。这里的已有应用包括遗留应用、预包装的应用和各种企业数据存储。在参考架构中，主要有两类访问服务。

（1）可接入服务（On-Ramp Service）：通过各种消息通信模式（单向、请求/应答和轮询）将业务逻辑和业务数据包装成企业服务总线可以访问的功能。

（2）事件发现服务（Event Detect Service）：提供事件通知服务将已有应用和数据中的变化通过事件框架发布到企业服务总线上。

2）整合新开发的应用——业务应用服务

同已有应用和数据类似，新开发的应用也作为重要的业务逻辑成为企业集成的目标。以服务为中心的企业集成通过业务应用服务（Business Application Service）实现新应用集成。一方面，业务应用服务帮助程序员开发可重用、可维护和灵活的业务逻辑组件；另一方面，它也提供运行时的集成对业务逻辑组件的自治管理。在参考架构中，有三类业务应用服务。

（1）组件服务（Component Service）：为可重用的组件提供应用的运行时容器管理服务，如对象持久化、组件安全管理和事务管理等。

（2）核心服务（Core Service）：提供运行时的服务，包括内存管理、对象实例化和对象池、性能管理和负载均衡、可用性管理等。

（3）接口服务（Interface Service）：提供和其他企业系统集成的接口，如其他企业应用，数据库、消息系统和管理框架。

3）整合客户和业务伙伴（B2C/B2B）——伙伴服务

以服务为中心的企业集成通过伙伴服务提供与企业外部的 B2B 的集成能力。因为业务伙伴系统的异构性，伙伴服务需要支持多种传输协议和数据格式。在参考架构中，提供如下服务。

（1）社区服务（Community Service）：用于管理和企业贸易的业务伙伴，支持以交易中心（Trade Hub）为主的集中式管理和以伙伴为中心的自我管理。

（2）文档服务（Document Service）：用于支持和业务伙伴交换的文档格式，以及交互的流程和状态管理，支持主流的 RosettaNet、EDI 和 AS1/AS2 等。

（3）协议服务（Protocol Service）：为文档的交互提供传输层的支持，包括认证和路由等。

3．控制服务

1）数据整合——信息服务

企业数据的分布性和异构性是应用系统方便访问企业数据和在企业数据之上提供增值服务的主要障碍。数据集成和聚合技术在这种背景下诞生，用于提供对分布式数据和异构数据的透明访问。

以服务为中心的企业集成通过信息服务提供集成数据的能力，目前主要包括如下集中信息服务。

（1）联邦服务（Federation Service）：提供将各种类型的数据聚合的能力，它既支持关系型数据，也支持像 XML 数据、文本数据和内容数据等非关系型数据。同时，所有的数据仍然按照自己本身的方式管理。

（2）复制服务（Replication Service）：提供远程数据的本地访问能力，它通过自动的实时复制和数据转换，在本地维护一个数据源的副本。本地数据和数据源在技术实现上可以是独立的。

（3）转换服务（Transformation Service）：用于数据源格式到目标格式的转换，可以是批量的或者是基于记录的。

（4）搜索服务（Search Service）：提供对企业数据的查询和检索服务，既支持数据库等结构化数据，也支持像 PDF 等非结构化数据。

2）流程整合——流程服务

企业部门内部的 IT 系统通过将业务活动自动化来提高业务活动的效率。但是这些部门的业务活动并不是独立的，而是和其他部门的活动彼此关联的。毋庸置疑，将彼此关联的业务活动组成自动化流程可以进一步提高业务活动的效率。业务流程集成正是在这一背景下诞生的。

以服务为中心的企业集成通过流程服务来完成业务流程集成。在业务流程集成中，粒度的业务逻辑被组合成业务流程，流程服务提供自动执行这些业务流程的能力。在参考架构中，流程服务包括如下内容。

（1）编排服务（Choreography Service）：通过预定义的流程逻辑控制流程中业务活动的执行，并帮助业务流程从错误中恢复。

（2）事务服务（Transaction Service）：用于保证流程执行中的事务特性（ACID）。对于短流程，通常采用传统的两阶段提交技术；对于长流程，一般采用补偿的方法。

（3）人工服务（Staff Service）：用于将人工的活动集成到流程中。一方面，它通过关联的交互服务使得人工可以参与到流程执行中；另一方面，它需要管理由于人工参与带来的管理任务，如任务分派，授权和监管等。

3）用户访问整合——交互服务

将适当的信息、在适当的时间、传递给适当的人一直是信息技术追求的目标。用户访问集成是实现这一目标的重要一环，它负责将信息系统中的信息传递给客户，不管它

在哪里，以什么样的设备接入。

以服务为中心的企业集成，通过交互服务来实现用户访问集成。参考架构中的交互服务包括如下类型。

（1）交付服务（Delivery Service）：提供运行时的交互框架，它通过各种技术支持同样的交互逻辑可以在多种方式（图形界面、语音和普及计算消息）和设备（桌面、PDA 和无线终端等）上运行，例如通过页面聚合和标签翻译使得同一个 Portlet 可以在桌面浏览器和 PDA 浏览器上展现。

（2）体验服务（Experience Service）：通过用户为中心的服务增强用户体验，其中的技术包括个性化、协作和单点登录等。

（3）资源服务（Resource Service）：提供运行时交互组件的管理，如安全配置、界面皮肤等。

4. 开发支持

企业集成涉及面很广，不仅需要开发新的应用并使其成为可以被用于企业集成的功能组件，而且需要将被包装的已有的应用和数据用于集成；不仅有企业内部的集成，而且需要和企业外部的系统集成；不仅有交互集成和数据集成，还有功能和应用集成。考虑到这其中的每部分在技术上都会涉及到各种平台和中间件，企业集成的技术复杂性是普通应用开发不可比拟的。这种技术复杂性需要更强有力的开发工具支持。企业集成的开发工具需要有标准的工具框架，这些工具能够以即插即用方式支持来自多家厂商的开发工具。同时，企业集成的开发工具需要支持整个软件开发周期，以提高开发过程中各种角色的生产力。

在以服务为中心的企业集成中，除了需要支持整个软件开发周期和标准的工具框架以外，开发服务需要提供和服务开发相关的技术。

（1）用于支持以服务为中心的企业集成方法学和建模，如 SODA 和 IBM 的 SOMA（Service Oriented Modeling and Architecture）。

（2）用于服务为中心的编程模型，如 WSDL、BPEL4WS、SCA 和 SDO 等。

开发环境和工具中为不同开发者的角色提供的功能被称为开发服务。根据开发过程中开发者角色和职责的不同，有如下 4 类服务。

（1）建模服务（Model Service）：用于构建可视化的业务流程模型。

（2）设计服务（Design Service）：根据业务模型，进一步分解为服务组件，设计服务用于设计和开发这些服务组件。

（3）实现服务（Implementation Service）：用于将设计和开发的服务组件部署到生产环境中。

（4）测试服务（Test Service）：支持服务组件的单元测试和系统的集成测试。

5. 业务创新和优化

一方面，以服务为中心的企业集成通过各种集成提高信息流转速度，从而提高生产

效率。另一方面，以服务为中心的企业集成也为业务创新和优化提供了支持平台——业务创新和优化服务。

业务创新和优化服务以业务性能管理（Business Process Management，BPM）技术为核心提供业务事件发布、收集和关键业务指标监控能力。具体而言，业务创新和优化服务由以下服务组成。

（1）公共事件框架服务（Common Event Infrastructure Service）：通过一个公共事件框架提供 IT 和业务事件的激发、存储和分类等。

（2）采集服务（Collection Service）：通过基于策略的过滤和相关性分析检测感兴趣的服务。

（3）监控服务（Monitoring Service）：通过事件与监控上下文间的映射，计算和管理业务流程的关键性能指标（Key Performance Indicators，KPI）。

业务创新和优化服务与开发服务是紧密相联的。在建模阶段被确定的业务流程的关键性能指标，被转为特别的事件标志构建到业务流程中，建模过程中的业务流程也被转换为用于监控服务的监控上下文。在业务流程执行过程中，这些事件标志激发的事件被公共事件框架服务截获，经过采集服务的过滤被传递给监控服务用于计算关键性能指标。关键性能指标作为重要的数据被用于重构或优化业务流程，这种迭代的方法使得业务流程处于不断的优化中。

6．管理支持

为业务流程和服务提供安全、高效和健康的运行环境，也是以服务为中心的企业集成重要的部分，它由 IT 服务管理来完成。IT 服务管理包括如下两部分。

（1）安全和目录服务（Security and Directory Service）：企业范围的用户、认证和授权管理，如单点登录（SSO）。

（2）系统管理和虚拟化服务（System Management and Virtualization Service）：用于管理服务器、存储、网络和其他 IT 资源。

IT 服务管理中相当一部分服务是面向软硬件管理的；而另外一部分服务，特别是安全和目录服务，以及操作系统和中间件管理，会通过企业服务总线和其他服务集成在一起，用于实现业务流程和服务的非功能性需求，如性能、可用性和安全性等。

20.4　SOA 主要技术和标准

Web 服务作为实现 SOA 中服务的最主要手段。首先来了解跟 Web Service 相关的标准，它们大多以 WS-作为名字的前缀，所以统称 WS-*。Web 服务最基本的协议包括 UDDI、WSDL 和 SOAP，通过它们，可以提供直接而又简单的 Web Service 支持，如图 20-3

所示。

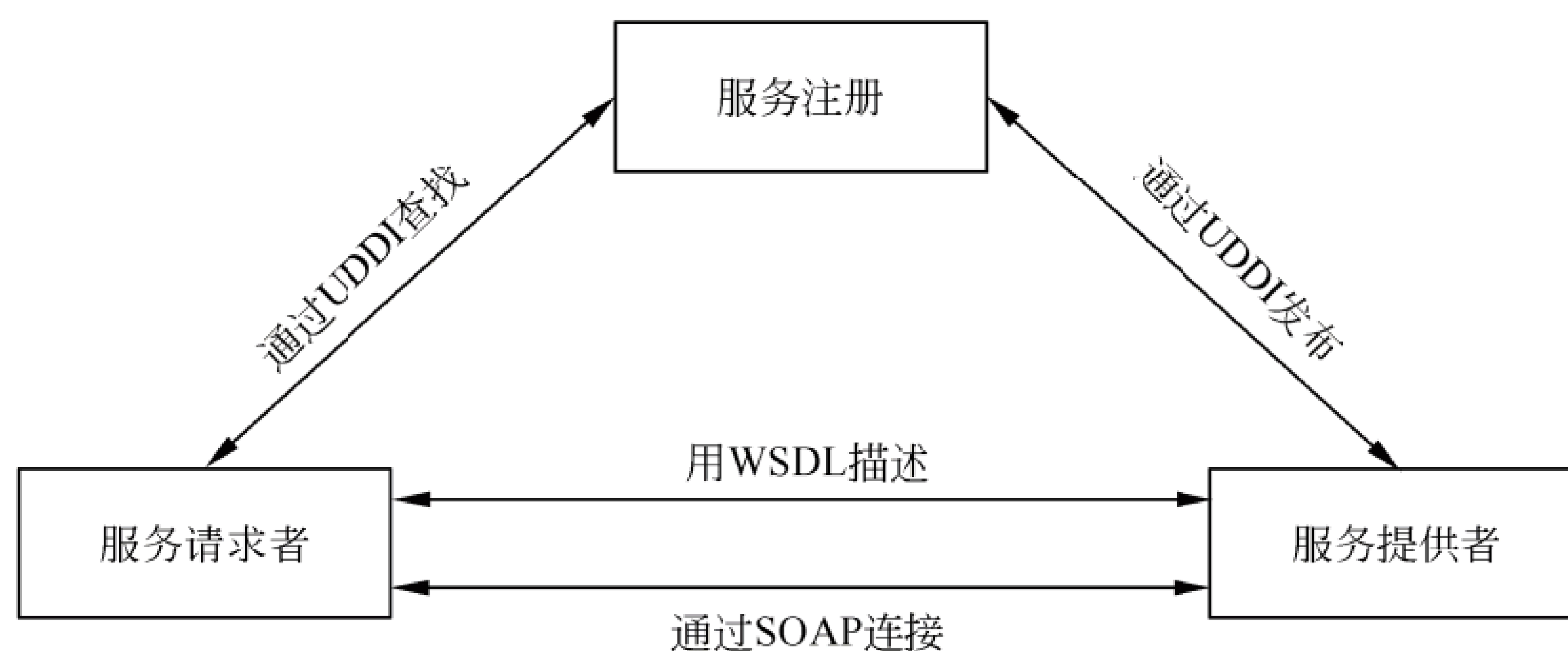


图 20-3　基本 Web 服务协议

20.4.1　UDDI 协议

UDDI（统一描述、发现和集成协议）计划是一个广泛的、开放的行业计划，它使得商业实体能够彼此发现；定义它们怎样在 Internet 上互相作用，并在一个全球的注册体系架构中共享信息。UDDI 是这样一种基础的系统构筑模块，它使商业实体能够快速、方便地使用它们自身的企业应用软件来发现合适的商业对等实体，并与其实施电子化的商业贸易。

UDDI 同时也是 Web 服务集成的一个体系框架，包含了服务描述与发现的标准规范。UDDI 规范利用了 W3C 和 Internet 工程任务组织的很多标准作为其实现基础，如 XML、HTTP 和 DNS 等协议。另外，在跨平台的设计特性中，UDDI 主要采用了已经被提议给 W3C 的 SOAP（Simple Object Access Protocol，简单对象访问协议）规范的早期版本。

20.4.2　WSDL 规范

WSDL（Web Services Description Language，Web 服务描述语言），是一个用来描述 Web 服务和说明如何与 Web 服务通信的 XML 语言。它是 Web 服务的接口定义语言，由 Ariba、Intel、IBM 和 MS 等共同提出，通过 WSDL，可描述 Web 服务的三个基本属性。

（1）服务做些什么——服务所提供的操作（方法）。

（2）如何访问服务——和服务交互的数据格式以及必要协议。

（3）服务位于何处——协议相关的地址，如 URL。

WSDL 文档以端口集合的形式来描述 Web 服务，WSDL 服务描述包含对一组操作和消息的一个抽象定义，绑定到这些操作和消息的一个具体协议，和这个绑定的一个网络端点规范。WSDL 文档被分为两种类型：服务接口（service interface）和服务实现（service implementations）。文档基本结构框架如图 20-4 所示。

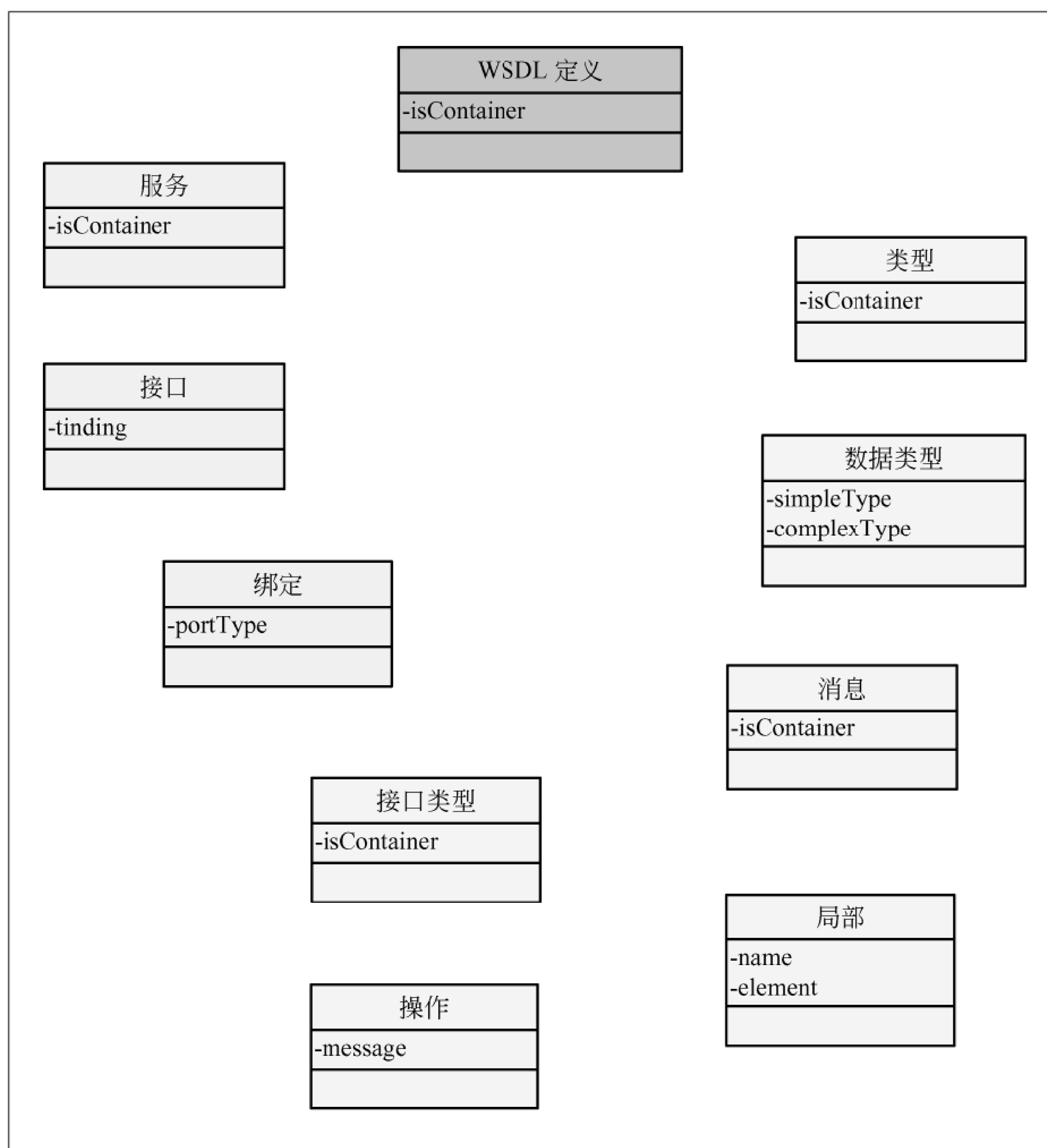


图 20-4　文档基本结构框架

服务接口文档中主要元素的作用分别如下。

- types：定义了 Web 服务使用的所有数据类型集合，可被元素的各消息部件所引用。它使用某种类型系统（一般使用 XML Schema 中的类型系统）。
- message：通信消息数据结构的抽象类型化定义。使用 Types 所定义的类型来定义整个消息的数据结构。
- operation：对服务中所支持操作的抽象描述。一般单个 operation 描述了一个访问入口的请求/响应消息对。
- portType：对于某个访问入口点类型所支持操作的抽象集合。这些操作可以由一个或多个服务访问点来支持。

- binding：包含了如何将抽象接口的元素（portType）转变为具体表示的细节，具体表示也就是指特定的数据格式和协议的结合；特定端口类型的具体协议和数据格式规范的绑定。
- port：定义为协议/数据格式绑定与具体 Web 访问地址组合的单个服务访问点。
- service：这是一个粗糙命名的元素，代表端口的集合；相关服务访问点的集合。

20.4.3 SOAP 协议

SOAP 是在分散或分布式的环境中交换信息的简单的协议，是一个基于 XML 的协议。它包括 4 个部分：SOAP 封装（Envelop），定义了一个描述消息中的内容是什么，是谁发送的，谁应当接收并处理它以及如何处理它们的框架；SOAP 编码规则（Encoding Rules），用于表示应用程序需要使用的数据类型的实例；SOAP RPC 表示（RPC Representation），表示远程过程调用和应答的协定；SOAP 绑定（Binding），使用底层协议交换信息。

虽然这 4 个部分都作为 SOAP 的一部分，作为一个整体定义的，但它们在功能上是相交的、彼此独立的。特别地，信封和编码规则是被定义在不同的 XML 命名空间（Namespace）中，这样使得定义更加简单。

SOAP 的两个主要设计目标是简单性和可扩展性，这就意味着有一些传统消息系统或分布式对象系统中的某些性质将不是 SOAP 规范的一部分。例如，分布式垃圾收集（Distributed Garbage Collection）、成批传送消息（Boxcarring or atching of messages）、对象引用 （Objects-by-reference which requires distributed garbage collection）和对象激活（Activation which requires objects-by-reference）等。

但是，基本协议无法保证企业计算需要的安全性和可靠性，所以需要增加这方面的协议，例如 WS-Security、WS-Reliability 和 WS-ReliableMessaging；对于复杂的业务场景，我们需要 WS-BPEL 和 WS-CDL 这样的语言来将多个服务编排成为业务流程；管理服务的协议如 WS-Manageability、WSDM 等。跟 Web 服务相关的标准，还在快速发展当中。目前在 SOA 产品和实践中，除了基本协议外，比较重要的还包括 BPEL、WS-Security、WS-Policy 和 SCA/SDO。

20.5 SOA 的特性

20.5.1 文档标准化

SOA 服务具有平台独立的自我描述 XML 文档。Web 服务描述语言是用于描述服务的标准语言。

20.5.2　通信协议标准

SOA 服务用消息进行通信，该消息通常使用 XML Schema 来定义（也叫做 XSD，XML Schema Definition）。消费者和提供者、或消费者和服务之间的通信多见于不知道提供者的环境中。服务间的通信也可以看作企业内部处理的关键商业文档。

20.5.3　应用程序统一登记与集成

在一个企业内部，SOA 服务通过一个扮演目录列表（Directory Listing）角色的登记处（Registry）来进行维护。应用程序在登记处（Registry）寻找并调用某项服务。统一描述、定义和集成是服务登记的标准。

20.5.4　服务品质

每项 SOA 服务都有一个与之相关的服务品质（Quality of Service，QoS）。QoS 的一些关键元素有安全需求（例如认证和授权）、可靠通信（译注：可靠消息是指确保消息"仅且仅仅"发送一次，从而过滤重复信息）以及谁能调用服务的策略。

在企业中，关键任务系统（Mission-critical System，译注：关键任务系统是指如果一个系统的可靠性对于一个组织是至关重要的，那么该系统就是该企业的关键任务系统。例如，电话系统对于一个电话促销企业来说就是关键任务系统，而文字处理系统就不那么关键了。）用来解决高级需求，例如安全性、可靠性和事务。当一个企业开始采用服务架构作为工具来进行开发和部署应用的时候，基本的 Web 服务规范，像 WSDL、SOAP 以及 UDDI 就不能满足这些高级需求。正如前面所提到的，这些需求也称作服务品质。与 QoS 相关的众多规范已经由一些标准化组织（Standards Bodies）提出，像 W3C 和 OASIS（the Organization for the Advancement of Structured Information Standards）。下面的部分将会讨论一些 QoS 服务和相关标准。

1．可靠性

在典型的 SOA 环境中，服务消费者和服务提供者之间会有几种不同的文档在进行交换。具有诸如"仅且仅仅传送一次（Once-and-only-once Delivery）"、"最多传送一次（At-most-once Delivery）"、"重复消息过滤（Duplicate Message Elimination）"和"保证消息传送（Guaranteed Message Delivery）"等特性消息的发送和确认，在关键任务系统（Mission-critical Systems）中变得十分重要。WS-Reliability 和 WS-ReliableMessaging 是两个用来解决此类问题的标准。这些标准现在都由 OASIS 负责。

2．安全性

Web 服务安全规范用来保证消息的安全性。该规范主要包括认证交换、消息完整性和消息保密。该规范吸引人的地方在于它借助现有的安全标准，例如，SAML（as Security Assertion Markup Language）实现 Web 服务消息的安全。OASIS 正致力于 Web 服务安全

规范的制定。

3．策略

服务提供者有时候会要求服务消费者与某种策略通信。例如，服务提供商可能会要求消费者提供 Kerberos 安全标示才能取得某项服务。这些要求被定义为策略断言（Policy Assertions），一项策略可能会包含多个断言。WS-Policy 用来标准化服务消费者和服务提供者之间的策略通信。

4．控制

在 SOA 中，进程是使用一组离散的服务创建的。BPEL4WS 或者 WSBPEL（Web Service Business Process Execution Language）是用来控制这些服务的语言。当企业着手于服务架构时，服务可以用来整合数据仓库（silos of data），应用程序，以及组件。整合应用意味着像异步通信，并行处理，数据转换，以及校正等进程请求必须被标准化。

5．管理

随着企业服务的增长，所使用的服务和业务进程的数量也随之增加，一个用来让系统管理员管理所有，运行在多种环境下的服务的管理系统就显得尤为重要。WSDM（Web Services for Distributed Management）的制定，使任何根据 WSDM 实现的服务都可以由一个 WSDM 适应（WSDM-compliant）的管理方案来管理。

其他的 QoS 特性，例如合作方之间的沟通和通信，多个服务之间的事务处理，都在 WS-Coordination 和 WS-Transaction 标准中描述，这些都是 OASIS 的工作。

20.6 SOA 的作用

在一个企业内部，可能存在不同的应用系统，而这些应用系统由于开发的时间不同，采用的开发工具不同，一个业务请求很难有效地调用所有的应用系统。用简单的语言来表述，这些已有应用系统是孤立的，也就是我们常说的“信息孤岛”。

不同种类的操作系统，应用软件，系统软件和应用基础结构（Application Infrastructure）相互交织，这是“信息孤岛”的表现症状。一些现存的应用程序被用来处理当前的业务流程（Business Processes），因此从头建立一个新的基础环境是不可能的。企业应该能对业务的变化做出快速的反应，利用对现有的应用程序和应用基础结构的投资来解决新的业务需求，为客户、商业伙伴以及供应商提供新的互动渠道，并呈现一个可以支持有机业务（Organic Business）的构架。SOA 凭借其松耦合的特性，使得企业可以按照模块化的方式来添加新服务或更新现有服务，以解决新的业务需要，提供选择从而可以通过不同的渠道提供服务，并可以把企业现有的或已有的应用作为服务，从而保护了现有的 IT 基础建设投资。

在 SOA 得以普及之前，解决企业内部信息系统“信息孤岛”的问题通常是采用 EAI（企业应用整合）的方式。为了保证所有的应用能够互通互用，每一个应用都需要一个

EAI Server 来对应。打个简单的比方，EAI Server 就好像一个“翻译”一样，让每两个应用之间可以对话，可以互相调用。但是，这样会带来 EAI Server 呈几何倍数的增长，当一个企业只有两个应用的时候需要一个“翻译”，当企业有三个应用需要互通的时候需要三个“翻译”，当有四个应用的时候就需要六个“翻译”，五个应用互通就需要十个“翻译”……这显然不是解决“信息孤岛”的妥善办法。

SOA 对于实现企业资源共享，打破“信息孤岛”的步骤如下。

（1）把应用和资源转换成服务。

（2）把这些服务变成标准的服务，形成资源的共享。

从这个意义上讲，SOA 不仅仅是一个技术，而是一个软件架构。企业的决策者只需要根据企业的策略来制定流程，把应用作为服务“拿来就用”，而无需考虑底层的集成。这样就可以实现 IT 和企业业务之间同步。

一个服装零售组织拥有 500 家国际连锁店，它们常常需要更改设计来赶上时尚的潮流。这可能意味着不仅需要更改样式和颜色，甚至还可能需要更换布料、制造商和可交付的产品。如果零售商和制造商之间的系统不兼容，那么从一个供应商到另一个供应商的更换可能就是一个非常复杂的软件流程。通过利用 WSDL 接口在操作方面的灵活性，每个公司都可以将它们的现有系统保持现状，而仅仅匹配 WSDL 接口并制订新的服务级协定，这样就不必完全重构它们的软件系统了。这是业务的水平改变，也就是说，它们改变的是合作伙伴，而所有的业务操作基本上都保持不变。这里，业务接口可以作少许改变。而内部操作却不需要改变。之所以这样做，仅仅是为了能够与外部合作伙伴一起工作。

20.7　SOA 设计原则

SOA 架构中，继承了来自对象和组件设计的各种原则，如封装、自我包含等。那些保证服务的灵活性、松散耦合和重用能力的设计原则，对 SOA 架构来说同样是非常重要的。

结构上，服务总线是 SOA 的架构模式之一。

关于服务，一些常见和讨论的设计原则如下。

（1）无状态。以避免服务请求者依赖于服务提供者的状态。

（2）单一实例。避免功能冗余。

（3）明确定义的接口。服务的接口由 WSDL 定义，用于指明服务的公共接口与其内部专用实现之间的界线。WS-Policy 用于描述服务规约，XML 模式（Schema）用于定义所交换的消息格式（即服务的公共数据）。使用者依赖服务规约调用服务，所以服务定义必须长时间稳定，一旦公布，不能随意更改；服务的定义应尽可能明确，减少使用者的不适当使用；不要让使用者看到服务内部的私有数据。

（4）自包含和模块化。服务封装了那些在业务上稳定、重复出现的活动和组件，实现服务的功能实体是完全独立自主的，独立进行部署、版本控制、自我管理和恢复。

（5）粗粒度。服务数量不应该太大，依靠消息交互而不是远程过程调用（RPC），通常消息量比较大，但是服务之间的交互频度较低。

（6）服务之间的松耦合性。服务使用者看到的是服务的接口，其位置、实现技术和当前状态等对使用者是不可见的，服务私有数据对服务使用者是不可见的。

（7）重用能力。服务应该是可以重用的。

（8）互操作性、兼容和策略声明。为了确保服务规约的全面和明确，策略成为一个越来越重要的方面。这可以是技术相关的内容，例如一个服务对安全性方面的要求；也可以是跟业务有关的语义方面的内容，例如需要满足的费用或者服务级别方面的要求，这些策略对于服务在交互时是非常重要的。WS-Policy 用于定义可配置的互操作语义，来描述特定服务的期望、控制其行为。在设计时，应该利用策略声明确保服务期望和语义兼容性方面的完整和明确。

20.8 SOA 的设计模式

20.8.1 服务注册表模式

服务注册表（Service Registry）主要在 SOA 设计时段使用，虽然它们常常也具有运行时段的功能。注册表支持驱动 SOA 治理的服务合同、策略和元数据的开发、发布和管理。因此，它们提供一个主控制点，或者称为策略执行点（Policy Enforcement Point，PEP）。在这个点上，服务可以在 SOA 中注册和被发现。

注册表可以包括有关服务和相关软件组件的配置、遵从性和约束配置文件。任何帮助注册、发现和检索服务合同、元数据和策略的信息库、数据库、目录或其他节点都可以被认为是一个注册表。

主要的服务注册厂商分为两个阵营。一个阵营是提供服务、策略和元数据注册表及信息库的纯 SOA 厂商，其中包括 Flashline、Infravio、LogicLibrary、SOA Software 和 Systinet（Mercury Interactive 下属分公司）；另一个阵营是 SOA 平台厂商，这些厂商将注册表作为集成产品套件的一个组件，他们的集成产品套件常常包括应用服务器、门户、数据库管理系统、BI 工具、集成中间件和其他功能组件。提供注册表的 SOA 平台厂商包括 BEA、IBM、Microsoft、Novell、Oracle、SAP、Sun 和 WebMethods。UDDI（通用描述、发现与集成）标准定义了 SOA 的一种主要注册环境，尽管这绝非唯一的环境。

大多数纯 SOA 厂商和 SOA 平台厂商还提供 SOA 开发、集成和管理工具。没有自己注册表的 SOA 厂商，常常通过 UDDI v3 和其他开放标准与一个或多个第三方注册表产品进行集成。大多数商用服务注册产品支持下面的 SOA 治理功能。

（1）服务注册：应用开发者，也叫服务提供者，向注册表公布他们的功能。他们公布服务合同，包括服务身份、位置、方法、绑定、配置、方案和策略等描述性属性。实现 SOA 治理最有效的方法之一，是限制哪类新服务可以向主注册表发布、由谁发布以及谁批准和根据什么条件批准。此外，许多注册表包含开发向注册表发布服务可能需要的说明性服务模板。

（2）服务位置：也就是服务应用开发者，帮助他们查询注册服务，寻找符合自身要求的服务。注册表让服务的消费者检索服务合同。对谁可以访问注册表，以及什么服务属性通过注册表暴露的控制，是另一些有效的 SOA 治理手段，注册表产品一般都支持此类功能。

（3）服务绑定：服务的消费者利用检索到的服务合同来开发代码，开发的代码将与注册的服务绑定、调用注册的服务以及与它们实现互动。开发者常常利用集成的开发环境自动将新开发的服务与不同的新协议、方案和程序间通信所需的其他接口绑在一起。工具驱动对服务绑定的控制，有效地管理服务在 ESB 上的互动。

设计时段，SOA 治理中新出现的最佳实践之一是注册表中的配置文件（Profile）管理。配置文件用于说明服务目前的生命周期阶段和该阶段的相关策略。Fiorano 公司的 CTO Atul Saini 是这样描述服务配置是如何在开发时段发挥作用的：“有人可能想在某台使用某个输入参数集合的机器上运行一项服务。机器名和参数成为与服务连接在一起的开发配置文件的一部分，一旦服务被开发，它可以被升级到质量保证阶段，运行在使用不同参数的不同机器上。第二台机器/参数集合构成一个新配置文件。这样，可以为某个服务创建多个配置文件，只需在任意时间将不同的配置文件与服务建立联系，这个服务就可以在其生命周期中的不同阶段之间移动。”

配置文件管理常常假设开发部门拥有一个将服务升级到下一阶段的结构化流程。一些 SOA 开发工具包含嵌入式工作流环境，帮助企业满足这方面的设计时段治理需要。LogicLibrary 公司 CTO、合作创始人 Brent Carlson 说：“公司的 Logidex 工具帮助开发部门将检查点、角色和多步骤工作流配置到 SOA 开发流程之中。”

他说：“您可以自动执行将服务提升到下一阶段所涉及的审查和验证，如果发现定义不一致，在服务向注册表发布之前，可将它退回开发者加以改正。”

20.8.2　企业服务总线模式

在企业基于 SOA 实施 EAI、B2B 和 BMP 的过程中，如果采用点对点的集成方式存在着复杂度高，可管理性差，复用度差和系统脆弱等问题。企业服务总线（Enterprise Service Bus，ESB）技术在这种背景下产生，其思想是提供一种标准的软件底层架构，各种程序组件能够以服务单元的方式“插入”到该平台上运行，并且组件之间能够以标准的消息通信方式来进行交互。它的定义通常如下：企业服务总线是由中间件技术实现的支持面向服务架构的基础软件平台，支持异构环境中的服务以基于消息和事件驱动模

式的交互，并且具有适当的服务质量和可管理性。

如图 20-5 所示，ESB 本质上是以中间件形式支持服务单元之间进行交互的软件平台。各种程序组件以标准的方式连接在该“总线”上，并且组件之间能够以格式统一的消息通信的方式来进行交互。一个典型的在 ESB 环境中组件之间的交互过程是：首先由服务请求者触发一次交互过程，产生一个服务请求消息，并将该消息按照 ESB 的要求标准化，然后标准化的消息被发送给服务总线。ESB 根据请求消息中的服务名或者接口名进行目的组件查找，将消息转发至目的组件，并最终将处理结果逆向返回给服务请求者。这种交互过程不再是点对点的直接交互模式，而是由事件驱动的消息交互模式。通过这种方式，ESB 最大限度上解耦了组件之间的依赖关系，降低了软件系统互连的复杂性。连接在总线上的组件无需了解其他组件和应用系统的位置及交互协议，只需要向服务总线发出请求，消息即可获得所需服务。服务总线事实上实现了组件和应用系统的位置透明和协议透明。技术人员可以通过开发符合 ESB 标准的组件（适配器）将外部应用连接至服务总线，实现与其他系统的互操作。同时，ESB 以中间件的方式，提供服务容错、负载均衡、QoS 保障和可管理功能。

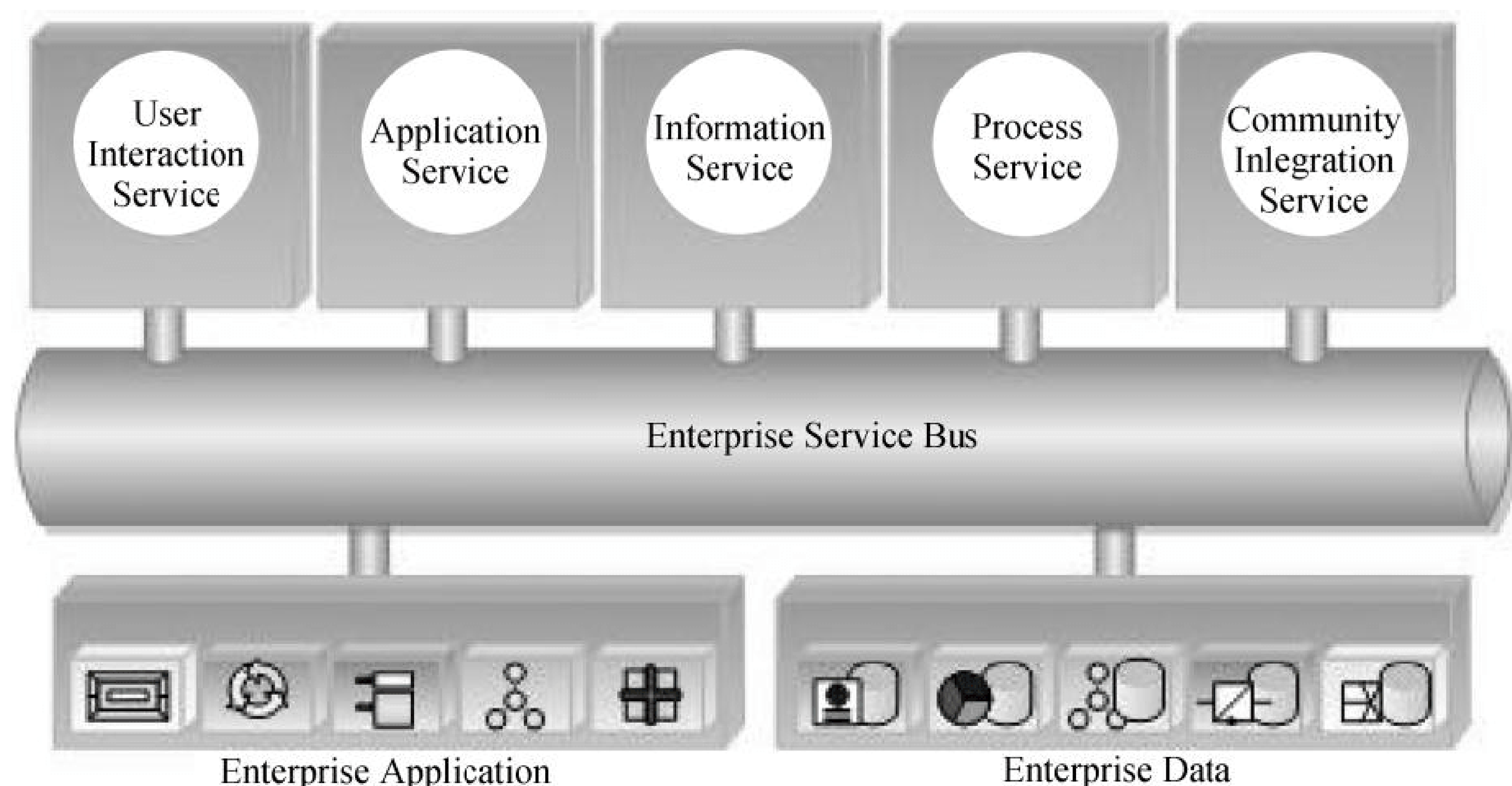


图 20-5　ESB 示意图

ESB 的核心功能如下。

（1）提供位置透明性的消息路由和寻址服务。

（2）提供服务注册和命名的管理功能。

（3）支持多种消息传递范型（如请求/响应、发布/订阅等）。

（4）支持多种可以广泛使用的传输协议。

（5）支持多种数据格式及其相互转换。

（6）提供日志和监控功能。

由于采用了基于标准的互连技术，ESB 使得企业内部以及外部系统之间可以很容易地进行异步或同步交互。它采用的面向服务的架构为系统提供了易扩展性和灵活性，在提高集成应用的开发效率的同时降低了成本。ESB 技术克服了传统应用集成技术的缺陷，能够对各种技术和应用系统提供支持，具有很强的灵活性和可扩展性，可以说是目前理想的 EAI、B2B 应用系统集成支撑平台。

ESB 本身为 EAI 提供了良好的支持平台，但是，作为最终的企业用户需要的则是包含业务集成软件基础平台、各种预制服务组件、集成应用开发、部署、管理和监控工具为一体的 EAI 环境。因此，作为软件厂商只是以 ESB 中间件作为基础软件平台，为用户提供整套立体的完善的企业应用软件集成平台。

案例研究

协同企业服务总线 SynchroESB 就是基于 SOA 体系结构，以 ESB 为底层架构，包含丰富的预制程序组件，集中式管理工具和可视化应用程序开发界面的服务整合软件平台。该产品在国家高新技术产业化计划的支持下，由西安协同时光软件公司和西北工业大学计算机学院联合研究开发的。系统结构如图 20-6 所示，系统分为 4 个层次设计。

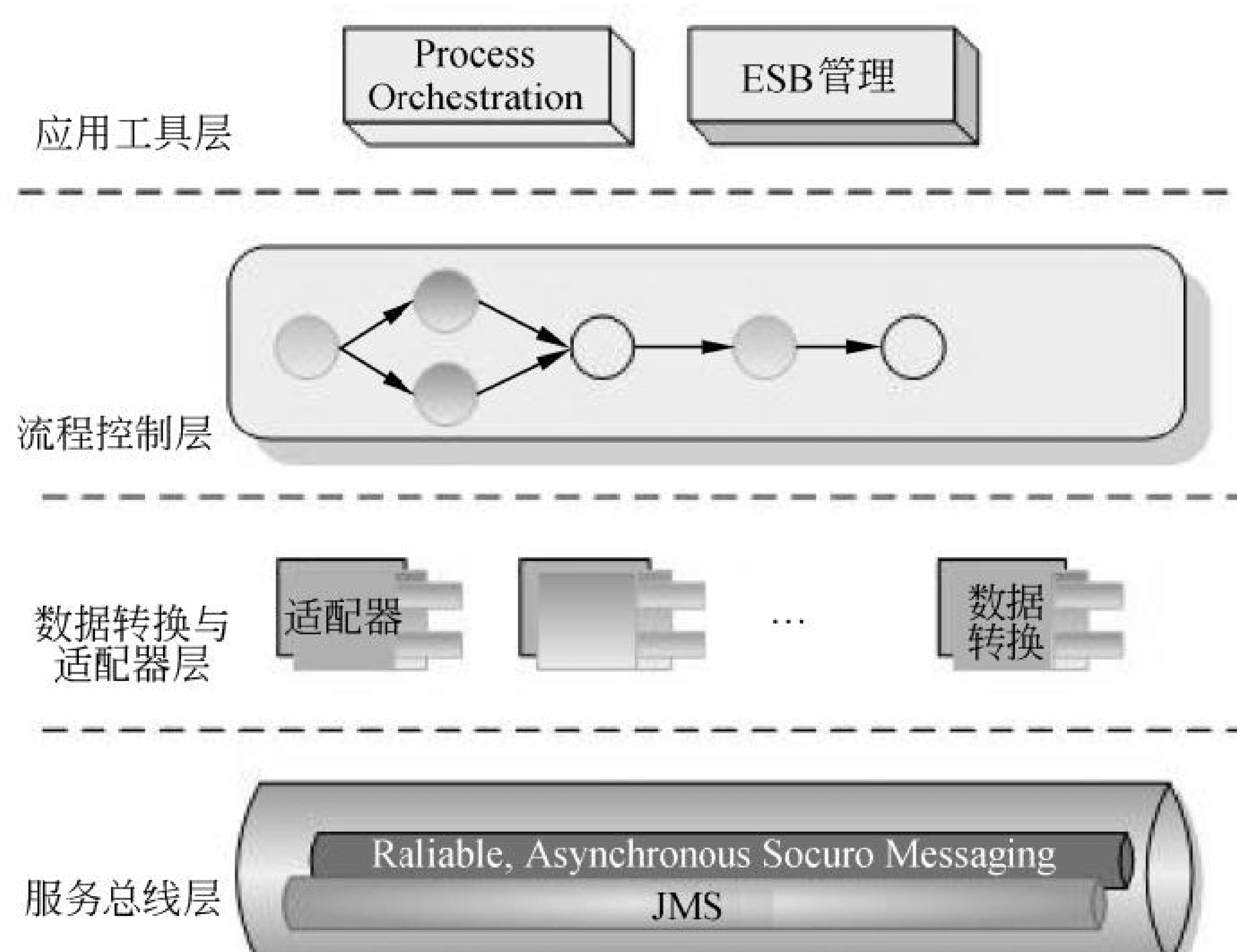


图 20-6　SynchroESB 层次结构图

服务总线层为整个 EAI 应用环境提供底层支持。ESB 层之上的数据转换与适配器层为各种 EAI 应用提供接入功能，它要解决的是应用集成服务器与被集成系统之间的连接和数据接口的问题。其上是流程整合层，它将不同的应用系统连接在一起，进行协同工作，并提供业务流程管理的相关功能，包括流程设计、监控和规划，实现业务流程的管理。最上端的用户交互层，则是为用户在界面上提供一个统一的信息服务功能入口，通过将内部和外部各种相对分散独立的信息组成一个统一的整体。

SynchroESB 支持企业构建可管理的、可扩展的和经济实用的 EAI 解决方案。它提供简单经济可扩展的方法和工具，以组件化的方式灵活构建业务流程。应用独创的“粗颗粒”组件编程模型技术构建可重用的组件库，使得诸如构建、原型化、生产和管理分布式复杂应用的活动，变得和今天我们习惯使用的电子表格操作一样简单。SynchroESB 支持企业以基于标准的、面向服务架构的方式将应用系统和流程跨越企业进行集成。通过分布式架构和集中式管理，SynchroESB 解决了集中式的集成方式中存在的问题，它使企业能够利用企业内任何地方的现有业务系统来快速组建一个有效的解决方案。SynchroESB 采用事件驱动架构使得企业能够更快地响应业务的变化。

20.9 构建 SOA 架构时应该注意的问题

20.9.1 原有系统架构中的集成需求

当架构师基于 SOA 来构建一个企业级的系统架构时，一定要注意对原有系统架构中的集成需求进行细致的分析和整理。我们都知道，面向服务的体系结构是当前及未来应用程序系统开发的重点。面向服务的体系结构本质上来说是一种具有特殊性质的体系结构，它由具有互操作性和位置透明的组件集成构建并互连而成。基于 SOA 的企业系统架构通常都是在现有系统架构投资的基础上发展起来的，我们并不需要彻底重新开发全部的子系统，SOA 可以通过利用当前系统已有的资源（开发人员、软件语言、硬件平台、数据库和应用程序）来重复利用系统中现有的系统和资源。SOA 是一种可适应的、灵活的体系结构类型，基于 SOA 构建的系统架构可以在系统的开发和维护中缩短产品上市时间，因而可以降低企业系统开发的成本和风险。因此，当 SOA 架构师遇到一个十分复杂的企业系统时，首先考虑的应该是如何重用已有的投资而不是替换遗留系统，因为如果考虑到有限的预算，整体系统替换的成本是十分高昂的。

当 SOA 架构师分析原有系统中的集成需求时，不应该只限定为基于组件构建的已有应用程序的集成，真正的集成比这要宽泛得多。在分析和评估一个已有系统体系结构的集成需求时，必须考虑一些更加具体的集成的类型，这主要包括以下几个方面：应用程序集成的需求，终端用户界面集成的需求，流程集成的需求以及已有系统信息集成的需求。当 SOA 架构师分析和评估现有系统中所有可能的集成需求时，可以发现实际上所有集成方式在任何种类的企业中都有一定程度的体现。针对不同的企业类型，这些集成方式可能是简化的，或者没有明确地进行定义的。因而，SOA 架构师在着手设计新的体系结构框架时，必须要全面地考虑所有可能的集成需求。例如，在一些类型的企业系统环境中可能只有很少的数据源类型，因此，系统中对消息集成的需求就可能会很简单。但在一些特定的系统中，例如航运系统中的 EDI（Electronic Data Interchange，电子数据交换）系统，会有大量的电子数据交换处理的需求，因此也就会存在很多不同的数据源

类型，在这种情况下整个系统对于消息数据的集成需求就会比较复杂。因此，如果 SOA 架构师希望所构建的系统架构能够随着企业的成长和变化成功地继续得以保持，则整个系统构架中的集成功能就应该由服务提供，而不是由特定的应用程序来完成。

20.9.2 服务粒度的控制以及无状态服务的设计

当 SOA 架构师构建一个企业级的 SOA 系统架构时，关于系统中最重要的元素，也就是 SOA 系统中服务的构建有两点需要特别注意的地方：首先是对于服务粒度的控制，另外就是对于无状态服务的设计。

1．服务粒度的控制

SOA 系统中服务粒度的控制是一项十分重要的设计任务。通常来说，对于将暴露在整个系统外部的服务推荐使用粗粒度的接口，而相对较细粒度的服务接口通常用于企业系统架构的内部。从技术上讲，粗粒度的服务接口可能是一个特定服务的完整执行，而细粒度的服务接口可能是实现这个粗粒度服务接口的具体的内部操作。举个例子来说，对于一个基于 SOA 架构的网上商店来说，粗粒度的服务可能就是暴露给外部用户使用的提交购买表单的操作，而系统内部细粒度的服务可能就是实现这个提交购买表单服务的一系列的内部服务，如创建购买记录、设置客户地址和更新数据库等一系列的操作。虽然细粒度的接口能为服务请求者提供更加细化和更多的灵活性，但同时也意味着引入较难控制的交互模式易变性，也就是说服务的交互模式可能随着不同的服务请求者而不同。如果我们暴露这些易于变化的服务接口给系统的外部用户，就可能造成外部服务请求者难于支持不断变化的服务提供者所暴露的细粒度服务接口；而粗粒度服务接口保证了服务请求者将以一致的方式使用系统中所暴露出的服务。虽然面向服务的体系结构并不强制要求一定要使用粗粒度的服务接口，但是建议使用它们作为外部集成的接口。通常架构设计师可以使用 BPEL 来创建由细粒度操作组成的业务流程的粗粒度的服务接口。

2．无状态服务的设计

SOA 系统架构中的具体服务应该都是独立的、自包含的请求，在实现这些服务的时候不需要前一个请求的状态，也就是说服务不应该依赖于其他服务的上下文和状态，即 SOA 架构中的服务应该是无状态的服务。当某一个服务需要依赖时，最好把它定义成具体的业务流程（BPEL）。在服务的具体实现机制上，可以通过使用 EJB 组件来实现粗粒度的服务。我们通常会利用无状态的 Session Bean 来实现具体的服务，如果基于 Web Service 技术，就可以将无状态的 Session Bean 暴露为外部用户可以调用到的 Web 服务，也就是把传统的 Session Facade 模型转化为 EJB 的 Web 服务端点。这样，就可以向 Web 服务客户提供粗粒度的服务。

如果要在 J2EE 的环境下（基于 WebSphere）构建 Web 服务，Web 服务客户可以通过两种方式访问 J2EE 应用程序。客户可以访问用 JAX-RPC API 创建的 Web 服务（使用

Servlet 来实现)；Web 服务客户也可以通过 EJB 的服务端点接口访问无状态的 Session Bean，但 Web 服务客户不能访问其他类型的企业 Bean，如有状态的 Session Bean、实体 Bean 和消息驱动 Bean。后一种选择（公开无状态 EJB 组件作为 Web 服务）有很多优势，基于已有的 EJB 组件，可以利用现有的业务逻辑和流程。在许多企业中，现有的业务逻辑可能已经使用 EJB 组件编写，通过 Web 服务公开它可能是实现从外界访问这些服务的最佳选择。EJB 端点是一种很好的选择，因为它使业务逻辑和端点位于同一层上。另外，EJB 容器会自动提供对并发的支持，作为无状态 Session Bean 实现的 EJB 服务端点不必担心多线程访问，因为 EJB 容器必须串行化对无状态会话 Bean 任何特定实例的请求。由于 EJB 容器都会提供对于 Security 和 Transaction 的支持，因此 Bean 的开发人员可以不需要编写安全代码以及事务处理代码。性能问题对于 Web 服务来说一直都是一个问题，由于几乎所有 EJB 容器都提供了对无状态会话 Bean 群集的支持以及对无状态 Session Bean 池与资源管理的支持，因此当负载增加时，可以向群集中增加机器。Web 服务请求可以定向到这些不同的服务器，同时由于无状态 Session Bean 池改进了资源利用和内存管理，使 Web 服务能够有效地响应多个客户请求。由此可以看到，通过把 Web 服务模型化为 EJB 端点，可以使服务具有更强的可伸缩性，并增强了系统整体的可靠性。

20.10 SOA 实施的过程

20.10.1 选择 SOA 解决方案

在实施 SOA 之前，选择最佳的解决方案，是保证 SOA 实施成功的前提条件。总体来说，必须从以下三个方面进行选择。

1．尽量选择能进行全局规划的方案

SOA 的实施，有很大的技术因素在其中，作为用户来讲，既需要选择适当的工具，还需要有专业的技术人才。

作为用户，实施 SOA，首先要对自己的系统做全面的评估，要了解自己已有的系统能用多少，有多少需要改造，还需要上哪些新的系统，自己将来的系统该如何满足自己的需求，自己可能为这个新的系统投入的资本大概有多少等。总之，要有整体的规划，这也是实施 SOA 最为基础的一步。其次，要选择适合的工具和技术。上什么系统，建什么平台，先改造哪个系统，需要一步一步来，而在这个过程中，所选择的产品也必然有所不同，一定要做到心中有数。最后，就是开发的过程了，开发对于大多数的用户来说，也是一个边学习、边实践的过程。

2．选择时充分考虑企业自身的需求

评估 SOA 项目的方式与评估传统软件项目有所不同，SOA 在企业范围内通过各种

渠道表现自己的优势。SOA 通过共享服务来优化业务流程，使全面创新成为可能，其“价值机会”远远超过了传统的软件项目。要建立强大的业务实例，通过 SOA 实现业务创新是一个重要的分水岭。必须认识到，用于构建 SOA 项目的前期投资将产生巨大效益，这些好处会随着时间的推移越来越明显地表现出来。

SOA 具体实施的进度和资金投入一方面取决于企业对 IT 应用的沉淀，另一方面取决于实行 SOA 的目标层次。

3．从平台、实施等技术方面进行考察

用户在选择 SOA 产品和技术时，应该从平台的选择、实施方法与途径、供应商的选择三个方面进行考量。在选择软件平台时，用户首先要考虑的是平台的开放性和对标准的支持。在实施方法与途径方面，以往的成功经验总结有 6 个方面：业务战略和流程、基础架构、构建模块、项目和应用、成本和效益以及规划和管理。在实施 SOA 时，CIO 应该综合考虑这 6 方面的因素。SOA 的实施涉及到整个企业的 IT 系统以及业务流程的调整和改变，离不开相应的咨询和专业服务。因此，在选择供应商时，首先要看它的产品是否符合企业的实际需求、是否已经有很多成功的应用案例、现有客户对它的评价如何；其次，还要仔细考察供应商的专业服务能力，是否能够帮助用户分析企业 IT 现状，提出建设性的意见。

20.10.2　业务流程分析

1．建立服务模型

1）自顶向下分解法

自上而下的领域分解方式从业务着手进行分析，选择端到端的业务流程进行逐层分解至业务活动，并对其间涉及的业务活动和业务对象进行变化分析。

业务组件模型是业务领域分解的输入之一。业务组件模型是一种业务咨询和转型的工具，它根据业务职责、职责间的关系等因素，将业务细分为业务领域、业务执行层次和业务组件。由于企业内部和外部环境的不同，每个业务组件在成本、投资和竞争力等方面不尽相同。因此，每个业务组件在企业发展的过程中战略职责和演化的路径也是不同的。由于角度的不同，就形成了所谓的业务组件的“热点视图”。对于面向服务的分析和设计，业务组件模型提供了进行服务划分的依据，而且这种划分的方法可以平滑地从业务视图细化到服务视图。

端到端的业务流程是业务领域分解的另一个输入。将业务流程分解成子流程或者业务活动，逐级进行，直到每个业务活动都是具备业务含义的最小单元。流程分解得到的业务活动树上的每一个节点，都是服务的候选者，构成了服务候选者组合。业务领域分解可以帮助发现主要的服务候选者，加上自下而上和中间对齐方式发现的新服务候选者，最终会构成一个服务候选者列表。在 SOA 的方法中，服务是业务组件间的契约，因此将服务候选者划分到业务组件，是服务分析中不可或缺的一步。服务候选者列表经过业

务组件的划分，会最终形成层次化的服务目录。

变化分析的目的是将业务领域中易变的部分和稳定的部分区分开来，通过将易变的业务逻辑及相关的业务规则剥离出来，保证未来的变化不会破坏现有设计，从而提升架构应对变化的能力。变化分析可能会从未来需求的分析中发现一些新的服务候选者，这些服务候选者需要加入到服务候选者目录中。

2）业务目标分析法

通过关键性能指标分析来验证已有服务候选者以及发现遗漏的服务候选者，这也可以叫做“目标服务建模（Goal Service Modeling）”。它的思想是这样的：从企业的业务目标出发，目标分解为子目标，子目标再分派给相关的服务来实现，这样就形成了一棵“目标服务树”，处于叶子节点上的每个服务都能回溯到具体的业务目标。第一步的工作必须基于之前对企业关键性能指标的分析之上。

3）自底向上分析法

自下而上方式的目的是利用已有资产来实现服务，已有资产包括已有系统、套装或定制应用、行业规范或业务模型等。这也可以叫做“遗留资产分析”，它的主要思想是：通过建立已有系统所具有的功能模块目录列表，可以方便地发现那些在不同的系统中被重复实现的功能模块以及可以复用的功能模块，从而将这些模块包装成服务发布出来。遗留资产分析的来源一般是原有系统的分析和设计文档，遗留系统分析的结果是可以重用的服务列表。

通过对已有资产的业务功能、技术平台、架构及实现方式的分析，除了能够验证服务候选者或者发现新的服务候选者，还能够通过分析已有系统、套装或定制应用的技术局限性，尽早验证服务实现决策的可行性，为服务实现决策提供重要的依据。

2．建立业务流程

1）建立业务对象

业务对象（Business Object，BO）是对数据进行检索和处理的组件，是简单的真实世界的软件抽象。业务对象通常位于中间层或者业务逻辑层。

业务对象可以在一个应用中自动地加入一个特定的功能来获得增值效应，使知识重用变为可能。例如，如果要开发一个包含多货币处理的应用，可以选择使用一个已经开发完成的，包含所有多货币处理功能的业务对象来开始你的开发，使开发工作极大地减少。

业务对象的分类如下。

（1）实体业务对象。表达了一个人、地点、事物或者概念。根据业务中的名词从业务域中提取，如客户、订单和物品。

（2）过程业务对象。表达应用程序中业务处理过程或者工作流程任务，通常依赖于实体业务对象，是业务的动词。

（3）事件业务对象。表达应用程序中由于系统的一些操作造成或产生的一些事件。

通过对业务对象的抽象，你的架构系统将体现更高的架构体系高度。

2）建立服务接口

在实现 SOA 解决方案的上下文中，服务接口的结构非常重要。设计糟糕的服务接口可能会极大地导致使用此接口的很多服务使用者应用程序的开发过程变得非常复杂。从业务角度而言，设计糟糕的服务接口可能使得业务流程的开发和优化变得复杂；相反，设计良好的服务接口可以加速开发计划的执行，并对业务级别的灵活性起到促进作用。

服务接口通常应该包含多个操作，定义为单个服务接口一部分的操作应该从语义上相关，仅包含单个操作或少量操作的大部分服务都表明服务粒度不恰当；反过来，采用很少的服务（或者单个服务）来包含大量操作也同样表明服务粒度不恰当。

服务之间的交换可以为有状态、也可以为无状态。当服务提供者保留关于在之前的操作调用期间服务使用者和服务提供者之间交换的数据信息时，服务之间进行的是有状态（或对话型）交换。例如，服务接口可以定义为 setCustomerNumber()和 getCustomerInfo()的操作。在有状态交换中，服务请求者将首先调用 setCustomerNumber()操作，并同时传入客户编号；服务提供者在内存中保留客户编号；接下来，服务请求者调用 getCustomer Info()操作，服务提供者将随后返回与之前调用中设置的客户编号对应的客户信息响应。

在构建 SOA 的过程中，将无状态接口视为最好的选择。无状态接口可以方便地供很多服务使用者应用程序重用，可以采用最适合每个应用程序的方式管理状态。传入操作的请求消息应该包含完成该操作所必要的所有信息，而不受到调用其他接口操作的顺序的影响。

3）建立业务流程

流程是指定的活动顺序，包含明确确定的用于提供业务值的输入和输出。例如，技术文档搜索流程从 Web 页面提取客户的搜索请求，并生成可选的文档列表。

对流程进行建模应当确保捕获的相关信息的一致性及完整性，以便业务分析员及开发人员能够理解模型所捕获的业务需求。在建模过程中，除了正常操作以外，标准流程的其他操作和异常必须获取，具有不同领域兴趣的专职人员和专家可以构建适合于大范围业务对象的流程模型。例如，分析员需要对流程有高度的见解以做出战略性决策，并进行诸如仿真之类的流程分析；开发人员将流程模型作为输入来实现解决方案。

分析员基于从业务需求所有者中所收集的需求构建业务流程（Business Process，BP）模型。通过使用适当的工具（例如 PowerPoint、spreadsheets、IBM Rational Requisite Pro 或者其他任意工具组合，并且在适当的时候可能是流程建模工具本身）来收集这些需求，分析员将这些需求及对现有流程的分析作为构建模型的输入条件，现有的流程模型用于对其进行分析或者通过修改现有的模型来创建新的流程模型，而不用从头重新创建。

通过将 BP 分成子流程开始建模过程。随后是对感兴趣的各子流程进行分析以确定组件、服务、输入输出数据、策略及测量。通过使用 WebSphere Business Integration Modeler 软件工具（Business Integration Modeler）将这些元素编码到 BP 模型中。

使用一种名为流程元素的建模构件来定义 BP 段，将其设计为可复用。流程元素是一种定义流程段的构件资产，在 BP 模型中，这种流程段被设计为可复用的构件来管理。它们将已建立的一系列任务、决策、对数据对象的引用、策略、角色及测试合并起来，例如，登录流程元素包含一系列活动，登录证书数据以及完成用户登录过程的登录规则。

这些流程元素表示可接受的操作行为，类似的需求也可复用它们。例如，作为子流程模型可检验并为购物篮中的商品定价。

BP 分析员与 BP 所有者及领域专家协作来获取所需的全部信息以构建 BP 模型。例如，分析员使用适当的工具收集角色、任务、序列信息、资源、数据、叙述和需求等，并将它们作为构建 BP 模型的输入内容。通过在 Business Integration Modeler 中创建流程模型，业务分析员所获取的信息可以轻易地导出给工作流开发人员，使他们在 Application Developer 工具中使用这些信息。

为流程建模的任务包括定义业务流程的细节，并为所有数据、资源及流程中所使用的其他元素建模。业务流程包含一些流程步骤，它们通过控制流相连接，这些控制流将活动与决策点相连。决策点遵循业务规则（转换条件），使用这些业务规则来确定流程应当依照什么路线进行。建模包括将 BP 分解成子流程并将所需的流程元素添加到模型中；分析员可以将现有的模型构件（例如，服务或流程元素）用于促进并加速模型的构建。

第 21 章　案 例 研 究

目前，软件体系架构技术依然是工业界和学术界探讨并不断发展的学科，属于起步阶段。工业界和学术界都用自己的方式表达对体系结构的概念与思维和探索。本章选择了一些案例或文章，以便于读者分析、研究体系结构。

21.1　价值驱动的体系结构：连接产品策略与体系结构

原文参见 URL（http://msdn2.microsoft.com/zh-cn/library/aa480060.aspx）。

系统的存在是为了为利益相关方创造价值。然而，这种理想往往无法完全实现。当前的开发方法给利益相关方、架构师和开发人员提供的信息是不完全和不充分的。这里介绍两个概念：价值模型和体系结构策略。它们似乎在许多开发过程中被遗忘，但创造定义完善的价值模型可以为提高折衷方案的质量提供指导，特别是那些部署到不同环境中的用户众多的系统。

21.1.1　价值模型概述

开发有目的的系统，其目的是为其利益相关者创造价值。在大多数情况下，这种价值被认为是有利的，因为这些利益相关者在其他系统中扮演着重要角色。同样，这些其他系统也是为了为其利益相关者创造价值。系统的这种递归特性是分析和了解价值流的一个关键。下一部分（发现价值模型）将对此进行更深入的讨论。

价值模型核心的特征可以简化为三种基本形式。

（1）价值期望值：表示对某一特定功能的需求，包括内容（功能）、满意度（质量）和不同级别质量的实用性。例如，汽车驾驶员对汽车从 60 英里每小时的速度进行急刹车的快慢和安全性有一种价值期望值。

（2）反作用力：系统部署实际环境中，实现某种价值期望值的难度，通常期望越高难度越大，即反作用力。例如，汽车从 60 英里每小时的速度进行紧急刹车的结果如何取决于路面类型、路面坡度和汽车重量等。

（3）变革催化剂：表示环境中导致价值期望值发生变化的某种事件，或者是导致不同结果的限制因素。

反作用力和变革催化剂称为限制因素，把这三个统称为价值驱动因素。如果系统旨在有效满足其利益相关者的价值模型要求，那么它就需要能够识别和分析价值模型。

传统方法，如用例方案和业务/营销需求，都是通过聚焦于与系统进行交互的参与者

的类型开始的。这种方法有如下几个突出的局限性。

（1）对参与者的行为模型关注较多，而对其中目标关注较少。

（2）往往将参与者固定化分成几种角色，其中每个角色所在的个体在本质上都是相同的（例如商人、投资经理或系统管理员）。

（3）往往忽略限制因素之间的差别（例如，纽约的证券交易员和伦敦的证券交易员是否相同？市场开放交易与每天交易是否相同？）。

（4）结果简单。要求得到满足或未得到满足，用例成功完成或未成功完成。

这种方法有一个非常合乎逻辑的实际原因。它使用顺序推理和分类逻辑，因此易于教授和讲解，并能生成一组易于验证的结果。

在《竞争优势》一书中，Michael Porter 以公司战略规划为背景讨论了价值链概念："虽然价值活动是竞争优势的构造块，但价值链并不是独立活动的一个集合，而是一系列相互依赖的活动的一个系统。联系是指一个价值活动的执行方式与另一个价值活动的性价比之间的关系"，"联系不仅存在于公司的价值链中（横向联系），还存在于公司价值链与供应商和渠道商价值链之间（纵向联系）。供应商供货渠道活动会影响公司活动的性价比（反之亦然）"。

效用曲线就是一个从一种度量标准到另外一种度量标准的映射。第一种度量标准表示一个可量化的结果变量，第二种度量标准是生成的值（满意度、效用）的级别。使用 Kepner 和 Tregoe 所描述的决策分析方法，各个可选方案会与各个期望值进行对照评估。效用曲线用于将每一个可选方案所得出的定量测量值映射到其对应值。然后，值级别用期望优先级加权，并进行叠加。叠加值越高，方案越可取。在某些情况下，该方法可能比较主观。

21.1.2 体系结构挑战

体系结构挑战是因为一个或多个限制因素使得满足一个或多个期望值变得更困难。在任何环境中，识别体系结构挑战都涉及评估。

（1）哪些限制因素影响一个或多个期望值？

（2）如果知道了影响，它们满足期望值更容易（积极影响）还是更难（消极影响）？

（3）各种影响的影响程度如何？在这种情况下，简单的低、中和高三个等级通常就已经够用了。

必须在体系结构挑战自己的背景中对其加以考虑。虽然跨背景平均效用曲线是可能的，但对于限制因素对期望值的影响不能采用同样的处理方法。例如，假设 Web 服务器在两种情况下提供页面。一种情况是访问静态信息，如参考文献。它们要求相应时间为 1～3 s。另一种情况是访问动态信息，如正在进行的体育项目的个人得分表。其响应时间为 3～6 s。

两种情况都有 CPU、内存、磁盘和网络局限性。不过，当请求量增加 10 或 100 倍

时，这两种情况可能遇到大不相同的可伸缩性障碍。对于动态内容，更新和访问的同步成为重负载下的一个限制因素。对于静态内容，重负载可以通过频繁缓存读页来克服。

制定系统的体系结构策略始于：

（1）识别合适的价值背景并对其进行优先化。

（2）在每一背景中定义效用曲线和优先化期望值。

（3）识别和分析每一背景中的反作用力和变革催化剂。

（4）检测限制因素使满足期望值变难的领域。

最早的体系结构决策产生最大价值才有意义。有几个标准可用于优先化体系结构。建议对以下几点进行权衡。

- 重要性：受挑战影响的期望值的优先级有多高？如果这些期望值是特定于不多的几个背景，那么这些背景的相对优先级如何？
- 程度：限制因素对期望值产生了多大影响？
- 后果：大概多少种方案可供选择？这些方案的难度或有效性是否有很大差异？
- 隔离：对最现实的方案的隔离情况如何？影响越广，该因素的重要性越高。

一旦体系结构挑战的优先级确定之后，就要确定处理最高优先级挑战的方法。尽管体系结构样式和模式技术非常有用，不过在该领域，在问题和解决方案领域的身后经验仍具有无法估量的价值。应对的有效方法源于技能、洞察力、奋斗和辛勤的工作。这个论断千真万确，不管问题是关于外科学、行政管理还是软件体系结构。

当制定了应对高优先级的方法之后，体系结构策略就可以表达出来了。架构是会分析这组方法，并给出一组关于以下领域的指导原则。

- 组织：如何将系统组织入子系统和组件？它们的组成和职责是什么？系统如何部署在网络上？都有哪些类型的用户和外部系统？它们位于何处？是如何连接的？
- 操作：组件如何交互？在哪些情况下通信是同步的？在哪些情况下是异步的？组件的各种操作是如何协调的？何时可以配置组件或在其上运行诊断？如何检测、诊断和纠正错误条件？
- 可变性：系统的哪些重要功能可以随部署环境的变化而变化？对于每一功能，哪些方案得到支持？何时可以做出选择（例如，编译、链接、安装、启动或在运行时）？各个分歧点之间有什么相关性？
- 演变：为了支持变更同时保持其稳定性，系统是如何设计的？哪些特定类型的重大变革已在预料之中，应对这些变更有哪些可取的方法？

总之，体系结构策略就是帆船的舵和龙骨，可以确定方向和稳定性。它应该是简短的高标准方向的陈述，必须能够被所有利益相关者所理解，并应在系统的整个生存期内保持相对稳定。

21.1.3 结论

价值模型有助于了解和传达关于价值来源的重要信息。它解决一些重要问题，如价值如何流动，期望值和外部因素中存在的相似性和区别，系统要实现这些价值有哪些子集。架构师分解系统产生一般影响的力，特定于某些背景的力和预计随着时间的推移而变化的力，以实现这些期望值。价值模型和软件体系结构的联系是明确而又合乎逻辑的，可以用以下 9 点来表述。

（1）软件密集型产品和系统的存在是为了提供价值。

（2）价值是一个标量，它融合了对边际效用理解和诸多不同目标之间的相对重要性。目标折衷是一个极其重要的问题。

（3）价值存在于多个层面，其中某些层面包含了目标系统，并将其作为一个价值提供者。用于这些领域的价值模型包含了软件体系结构的主要驱动因素。

（4）该层次结构中高于上述层面的价值模型可以导致其下层价值模型发生变化。这是制定系统演化原则的一个重要依据。

（5）对于每一个价值群，价值模型都是同类的。暴露于不同环境条件的价值背景具有不同的期望值。

（6）对于满足不同价值背景需要，系统的开发赞助商有着不同的优先级。

（7）体系结构挑战是由环境因素自某一背景中对期望的影响引起的。

（8）体系结构方法试图通过首先克服最高优先级体系结构挑战来实现价值的最大化。

（9）体系结构策略是通过总结共同规则、政策和组织原则、操作、变化和演变从最高优先级体系结构方法综合得出的。

21.2 使用 RUP 和 UML 开发联邦企业体系结构框架

原文参见 URL（http://www.ibm.com/developerworks/cn/rational/r-feaf/）。

对于贯彻联邦企业体系结构框架（Federal Enterprise Architecture Framework，FEAF）方针的团体和机构而言，IBM Rational Unified Process（RUP）是足以支持其企业体系结构（Enterprise Architecture，EA）计划的一种选择。本文探讨如何使用 RUP 和 UML 构建和管理企业体系结构。具体而言，将分析 FEAF 的 4 层矩阵结构（level IV matrix），并讨论如何用 RUP 促进捕获各种 FEAF 模型。

1996 年的克林格-科恩法案（Clinger-Cohen Act）授权联邦机构开发和维护一种企业 IT 体系结构，以便促进联邦机构间的信息共享和组织。1999 年，联邦首席信息官负责根据这一授权建立 FEAF，具体内容参阅 http://www.cio.gov/documents/fedarch1.pdf。FEAF 的目的是建立机构范围内的路线图，通过在有效的信息技术环境中优化其核心业务过程

的性能来履行机构的使命。企业体系结构可以帮助机构实现这一目标，简单地讲，它们系统而完整地定义了组织的当前（基准）环境和期望（目标）环境的蓝图。对于信息系统的演进以及开发优化其职能价值的新系统而言，EA是必不可少的。企业体系结构是从逻辑或业务（如职能、业务职责、信息流和系统环境）以及技术（如软件、硬件、通信）两方面来定义的，并且包括从基准环境转换到目标环境的顺序规划（Sequencing Plan）。

21.2.1　联邦企业体系结构框架概述

联邦企业体系结构框架作为一种组织机制，用于管理体系结构描述的开发和维护。FEAF 也提供了组织联邦资源、描述和管理联邦企业体系结构活动的结构。框架是通过把企业信息组织到不同的层次或参考结构中来实现这一目标的。最上面的第一层是企业的最高层视图，最下面的第 4 层包含最详细的企业信息。它把企业体系结构划分为 4 部分：业务、数据、应用程序和技术。FEAF 还考虑了 Zachman Framework 的元素以及 Spewak EA 规划方法学的应用。

FEAF 确定了开发和维护联邦企业体系结构所需的 8 种构件。这 8 种构件的分解进一步细化了 FEAF 的 4 个层次。前三层阐述了这 8 种构件逐步细化的过程，最终在第四层形成了分类和组织联邦企业描述性表示的结构。

第一层是联邦企业体系结构框架的最高层，它引入了开发和维护联邦企业体系结构所需要的 8 种构件。

- 体系结构推动者（Architecture Drivers）：代表推动联邦企业体系结构变更的外部激励因素。
- 战略方向（Strategic Direction）：确保变更和政府的总体方向一致。
- 当前体系结构（Current Architecture）：表示企业或机构的当前状态。完整描述可能非常重要，应该小心维护。
- 目标体系结构（Target Architecture）：表示战略方向环境中企业的目标状态。
- 转换过程（Transitional Processes）：这些过程按照体系结构标准施行从当前体系结构到目标体系结构的变更，如不同的决策或管理过程、迁移规划、预算、配置管理和变更控制。
- 体系结构片段（Architectural Segments）：关注整个企业中的某个子集或较小的企业。
- 体系结构模型（Architectural Models）：提供在企业中管理和实现变更的文档和基础。
- 标准（Standards）：机构所采用的标准（无论是强制采用还是自愿采用的），包括最佳实践和各种开放标准，所有标准都是为了提高互操作性。

第二层在更详细的层次上说明了联邦企业体系结构的业务和设计方面以及两者之

间的关联。业务体系结构和设计体系结构之间的关系是推/拉关系——业务推动设计以满足自身的需要，设计（即新开发的数据、应用程序和技术）通过支持业务运作来拉动业务到新的服务交付水平。

第一层所描述的 8 种元素在第二层中进一步细化，在更小的粒度上描述业务和设计。例如，在第二层中观察当前体系结构（Current Architecture）构件时，将关注当前业务体系结构（Current Business Architecture），它确定了当前设计所支持的当前业务需求。以及当前设计体系结构（Current Design Architectures），它定义了用于支持当前业务需求的当前实现的数据、应用程序和技术。对于第二层中的其他构件也可从类似的视角来观察。

第三层展开了框架的设计部分，显示三种设计体系结构：数据、应用程序和技术。设计体系结构进一步细化了第二层中列出的设计细节。下面是第三层中进一步细化的 6 种构件中的三种。

- 当前设计体系结构（Current Design Architectures）：用于支持当前业务需求已实现的设计。当前设计体系结构由数据体系结构、应用程序体系结构和技术体系结构组成。
- 目标设计体系结构（Target Design Architectures）：用于支持未来业务需求的未来设计。目标设计体系结构由目标数据体系结构、目标应用程序体系结构和目标技术体系结构组成。
- 设计模型（Design Models）：用于定义企业的模型。有数据模型、应用程序模型和技术模型三种类型。

第三层还提供了体系结构片段（Architectural Segment）、转换过程（Transitional Processes）和标准（Standards）这三种构件的更多细节。

第四层（最详细的视图）确定了描述业务体系结构和三种设计体系结构（数据、应用程序和技术）的模型种类。它还定义了企业体系结构规划。在第四层上，三种设计体系结构如何支持业务体系结构开始逐渐明确起来。在这一层上，FEAF 确定了两种机制：FEAF 矩阵和企业体系结构规划（Enterprise Architecture Planning，EAP）方法学。FEAF 矩阵用于组织体系结构信息，EAP 帮助定义什么样的体系结构适合特定的企业。

21.2.2 FEAF 矩阵概述

FEAF 提供了开发、维护和实现高层操作环境并支持 IT 系统实现的结构。这种结构根据 Zachman 框架来分类和组织企业的重要模型。Zachman Framework 是 1987 年由 John Zachman 提出的，是企业根据总体信息需求评估软件开发过程模型完整性的一种方法。该框架为完整的体系结构提供了多种视角，并对体系结构产品进行了分类。Zachman Framework 实际上是一个包括 36 个单元的矩阵，涵盖了企业中的谁（who）、什么（what）、何处（where）、何时（when）、为何（why）以及如何（how）。该框架把企业分解成 6 个视角（perspective），从最高层的业务抽象开始直到实现。该框架可以包含全局规划，

也可以包含技术细节、列表和图表。任何适当的步骤、标准、角色、方法或技术都可以放进去。

FEAF 关注 Zachman Framework 中的三个方面：数据（什么）、过程或应用程序（如何）和位置或技术（何处）。如图所示，把 FEAF 图形化表示为一个 3×5 的矩阵，体系结构类型（数据、应用程序和技术）是矩阵的一个轴，视角（规划者（Planner）、所有者（Owner）、设计者（Designer）、构建者（Builder）和转包者（Subcontractor））在另一个轴上。相应的 EA 产品列在矩阵的单元中。

图 21-1 提供了 FEAF 矩阵的综览，在第四层上描述了 FEAF。该矩阵结合了 5 个视角行（即视图）：规划者、所有者、设计者、构建者和转包者，以及 Zachman Framework 中的前三个体系结构工件或产品抽象列（即什么、如何和何处）。FEAF 矩阵也把视角或行称为视图，表示不同的抽象层次。此外，视角和焦点（列）的相交称为 FEAF 的“模型”。IBM Rational Unified Process 也结合最佳实践为不同的项目干系人和需求提供不同的抽象层次。在 RUP 中，体系结构通过不同的视图来定义，每个视图都依赖于特定项目干系人所需要的详细程度。关键体系结构决策在每个视图中表示。RUP 中的模型记录了所有做出的决策，包括体系结构上的重要决策。例如，用例模型可能包括 25 个用例，其中只有 10 个对体系结构非常重要。用例视图就仅仅表示对体系结构至关重要的那些用例。对于本文而言，FEAF 模型和 RUP 体系结构视图是等价的。此外，RUP 提供了一组一致的模型，把不同视图中的体系结构元素关联在一起。

	数据体系结构	应用程序体系结构	技术体系结构
规划者视角	业务对象列表	业务过程列表	业务位置列表
所有者视角	语义模型	业务过程模型	业务物流模型
设计者视角	逻辑数据模型	应用程序体系结构	系统地理部署体系结构
构建者视角	物理数据模型	系统设计	技术体系结构
转包者视角	数据词典	程序	网络体系结构

图 21-1　HL7 消息发展体系（引自 HL7 V3 Ballot 7）

规划者和所有者这两行关注的是业务体系结构的定义和编档。这两行一旦完成，就明确了企业的业务是什么，以及用什么样的信息来控制它（即业务模型）。这两行被认为是基础，要开发能够共同理解并跨联邦企业集成的体系结构描述必须要完成这两行。

第三、四、五行（即设计者、构建者和转包者）定义了支持业务体系结构的设计体系结构（即数据、应用程序和技术）。根据特定体系结构的用途和目标开发这几行的适当模型。

每个视角和设计体系结构的相交（Intersection）所定义的模型，是及时管理和实现企业变更的基础。对于那些支持系统管理和开发至关重要的企业模型，该框架提供了分

类和组织这种企业模型的逻辑结构。

21.2.3 使用 RUP 支持 FEAF

Rational Unified Process 主要强调的是软件系统。企业体系结构包括软件，但是还涉及到硬件、人员和信息。从 FEAF 强调数据、应用程序和技术设计体系结构可以看出这一点。本质上，企业组织可以看作一个包含其他系统的系统。虽然 RUP 确实讨论了如何表示软件应用程序的硬件、人员和信息，但是在解决系统问题时还有待于改进。为满足这种需要，RUP for System Engineering 出现了，它是一个 RUP 插件，提供了新的和改进的活动和工件，增强了 RUP 的功能。它还提供了一组技术用于减少功能分解的必要性，从而使系统和子系统规格说明满足整个开发团队的需要。本文不再深入探讨如何在 EA 开发中使用 RUP SE 技术，但是讨论了构建 EA 时使用的 RUP 和 RUP for System Engineering 工作流的详细情况。

表 21-1 说明了构造 FEAF 矩阵中各种模型（或者 RUP 体系结构视图）时应该使用 RUP 和 RUP for System Engineering 的哪一部分。下面的矩阵简要定义了要捕获的体系结构视图，如何使用 RUP 和 UML 捕获这些视图，以及有关使用 RUP 的更多信息的 RUP 工作流和活动参考。体系结构视图不是互相独立的，而是一组一致且可实现的模型视图。

表 21-1 FEAF 矩阵中的各种模型描述

视　角	数据体系结构 （实体=什么）	应用程序体系结构 （活动=如何）	技术体系结构 （位置=何处）
规划者 （作用域）	业务对象列表 定义：企业所关心的业务对象（或事物、资产）的高层列表。该模型定义了后续企业对象模型的作用域	业务过程列表 定义：企业执行过程的高层列表。该模型定义了后续企业过程模型的作用域	业务位置列表 定义：企业运作位置的高层列表。该模型定义了与企业关联的后续位置模型的作用域
	IBM Rational 方法： RUP 业务建模准则用于创建领域模型，强调解释业务领域中重要的“事物”和产品。在某种意义上，它创建了一个数据词典来捕获所有业务对象，作为使用或重用的建模元素。 这些对象可以使用 UML 作为简单对象或者没有关系的类图来捕获，如果需要可以生成文档	IBM Rational 方法： 业务建模是 Rational Unified Process 中的一条重要准则。该准则描述了如何开发组织的场景或任务陈述，在业务用例模型和业务对象模型中定义组织的过程、角色和职责。 业务过程列表可用 UML 的业务用例图表示。业务用例是业务执行的一系列动作，可以为特定业务参与者生成有价值的、可观测的结果	IBM Rational 方法： 业务位置列表被捕获和表示为一组地点，它在 RUP SE 中定义。地点表示处理发生的假想位置，不要试图对应特定的位置或者硬件。地点图用 UML 部署图来描述，其中的结点被构造为地点。这个特定视图中不一定需要地点之间的连接，列表可以从报告的模型中生成

续表

视　角	数据体系结构（实体=什么）	应用程序体系结构（活动=如何）	技术体系结构（位置=何处）
规划者（作用域）	RUP 参考： 更多信息请参阅“业务建模准则：开发领域模型工作流详解”	RUP 参考： 更多信息请参阅“业务建模准则：描述当前业务工作流详解”	RUP for Systems Engineering 参考： 更多信息请参阅“分析与设计准则：综合系统体系结构工作流详解”
所有者（企业）	语义模型（Semantic Model） 定义：是对于企业至关重要的实际企业业务对象（即事物、资产）的模型	业务过程模型（Business Process Model） 定义：表示企业执行的实际业务过程，不依赖于任何系统或者实现因素和组织性约束	业务物流系统（Business Logistics） 定义：捕获了企业的位置及它们之间的联系（即语音、数据、邮件或卡车、铁路及轮船等）。它确定了分支机构、总部、仓库等结点的各种类型设施
	IBM Rational 方法： 语义模型基本上是规划者视角中对象列表的精化。所有者视角精化了领域模型，把业务对象之间的关系包括进来。语义模型可以使用相同类型的 UML 图来捕获	IBM Rational 方法： 本单元中进一步分析上面确定的业务过程。使用 UML 活动图或者顺序图对不同工作者执行的事件或任务流建模。 顺序图或者活动图中包括的元素反映了如何协调企业的各种资源实现业务用例目标。元素应该是人员、应用程序、硬件和数据的组合。除了可视化的 UML 模型外，业务用例为进一步理解业务过程提供了文本性规格说明	IBM Rational 方法： 进一步精化了规划者视角中的地点，增加了连接信息。使用地点图说明各个不同的位置及它们的连接。连接线上的注解说明如何实现连接（即语音、数据、邮件或卡车、铁路及轮船等）。除了内部结点视角外，还可以用描述每个结点上设施的地点图完成
	RUP 参考： 更多信息请参阅“业务建模准则：开发领域模型工作流详解”	RUP 参考： 更多信息请参阅“业务建模准则：精化业务过程定义工作流详解”	RUP for Systems Engineering 参考： 更多信息请参阅“分析与设计准则：综合系统体系结构工作流详解”
设计者（信息系统）	逻辑数据模型（Logical Data Model） 定义：是记录企业信息的对象的逻辑表示。它用属性完备的、重要的、规格化的实体关系模型表示，反映了语义模型的意图	应用程序体系结构（Application Architecture） 定义：表示支持业务过程的逻辑系统实现。它表示系统中的人机边界	系统地理部署体系结构（System Geographic Deployment Architecture） 定义：是描述业务物流系统的系统实现的逻辑模型。它描述了结点上的设施类型和控制软件（应用程序），以及结点之间的通信线路（如处理器、操作系统、存储设备、DBMS 和外围设备/驱动程序）

续表

视　　角	数据体系结构 （实体=什么）	应用程序体系结构 （活动=如何）	技术体系结构 （位置=何处）
	IBM Rational 方法： 逻辑数据模型通过进一步精化语义模型来捕获。UML 类图用于进一步精化上面的语义模型。逻辑数据模型类图显示了数据实体、实体之间的关系，以及数据实体的属性和指定键	IBM Rational 方法： 现在，应用程序体系结构开发支持业务过程的单个应用程序或者系统的体系结构。应用程序体系结构中提出的工件对体系结构具有重要的影响	
设计者（信息系统）	RUP 参考： 更多信息请参阅“RUP 分析与设计准则：分析行为和数据库设计工作流详解”	RUP 在不同的准则和活动中为开发应用程序体系结构提供了指南，尤其是需求和分析设计准则。应用程序体系结构包括系统用例和相应的分析实现。分析实现为应用程序元素间的交互和关系提供了高层描述。交互和关系使用 UML 交互图（顺序图或协作图）和类图描述。这些实现在系统设计中进一步开发和详述	IBM Rational 方法： 该模型中现在开始定义从其他视图中各种细节派生出来的构件。地点用一组构件来实现，包括硬件、软件（应用程序）或人员。构件用节点来描述，节点被构造型化（stereotyped）为描述符节点，在 UML 部署图中查看
		RUP 参考： 更多信息请参阅“需求准则：定义系统和精化系统定义工作流详解”、“分析与设计准则：定义候选体系结构和分析行为工作流详解”	RUP for Systems Engineering 参考： 更多信息请参阅“分析与设计准则：综合系统体系结构工作流详解”
	物理数据模型 定义：表示已经精化到能够用于实际数据库实现的数据模型。物理数据模型描述支持逻辑模型所必需的结构，依赖于所选择的方法	系统设计 定义：定义了方法及其实现	技术体系结构 定义：是企业技术环境的物理表示。它说明了节点和连线上实际存在的硬件和软件系统，包括操作系统和中间件
构建者（技术）	IBM Rational 方法： 物理数据模型的建立把逻辑数据实体和属性映射到物理表和列。因为 UML 支持这种映射，所以只需要一种建模语言。物理数据模型	IBM Rational 方法： 系统设计进一步发展了应用程序体系结构中的分析实现，为实现提供所有必要的细节。RUP 在“分析与设计准则：明确的用例设计”、“子系统设计”、“类设计”等活	IBM Rational 方法： 技术体系结构描述了企业中将用于实现系统的实际物理硬件。它还表示了系统设计中分配在硬件上的软件系统。RUP 为如何在

续表

视　　角	数据体系结构（实体=什么）	应用程序体系结构（活动=如何）	技术体系结构（位置=何处）
构建者（技术）	使用 UML 数据建模概要文件表示。RUP 可以非常灵活地对物理数据模型建模。关系模型可以使用数据建模的UML概要文件捕获，面向对象的数据存储也可以使用属性完备的类图来捕获。此外，XML 模式也可使用 UML 建模	动中提供了捕获系统设计的详细指南。产品用顺序图和/或协作图描述设计元素间的动态交互，用类图表示对体系结构至关重要的设计类，用状态机表示具有重要状态行为的类，用构件图表示对体系结构很重要的软件组件	UML 部署图中捕获这些活动提供了指南
	RUP 参考： RUP“分析与设计准则：数据库设计”可用于此。	RUP 参考：RUP“分析与设计准则：精化体系结构和设计构件工作流详解”适用于这一活动	RUP 参考： RUP“分析与设计准则：精化体系结构工作流详解”适用于该活动
转包者（详细规格说明）	数据定义 定义：物理模型中规定的所有数据对象的定义，应该包括实现所需要的所有数据定义语言	程序 定义：实现了系统设计的应用程序实现	网络体系结构 定义：包括节点地址的具体定义和连线标识
	IBM Rational 方法： 数据定义是物理模型的实际实现。UML 规范可以直接转化成实现（DDL或者直接到数据库管理系统）。实现经常由物理模型自动生成	IBM Rational 方法： 系统设计中的每个元素都通过编码或者使用原有的构件得以实现。设计中的每个元素具体对应什么依赖于编程语言。用于系统设计的 UML 规格说明可以转化成各种编程语言，包括 Java、Visual Basic、C++、C#和 XML 等。此外，还可以采用模式帮助确保实现的一致性。模式凝聚了从实践中收集的特定知识。无论是自己发现的还是借用别人的，模式都提供了解决实际问题的建模范例	IBM Rational 方法： 网络体系结构是技术体系结构 UML 部署图的精化，说明具体的地址和连线标识

21.2.4　结论

建立和管理企业体系结构所需要的业务模型和设计模型可以使用不同的技术和方法来完成。IBM Rational Unified Process 提供了建立和维护企业体系结构的一组关联的最佳实践和方法。Rational Unified Process 把不同的视角和一组实践活动，以及创建一组

一致的模型所得到的工件结合在一起。模型的体系结构视图可以组织成 FEAF 矩阵。总之，使用 RUP 作为开发企业体系结构的过程框架，组织可以有效地捕获、审查、管理变更，并可在不同视角和组织之间沟通企业体系结构。

21.3 Web 服务在 HL7 上的应用——Web 服务基础实现框架

原文参见 URL（http://msdn2.microsoft.com/zh-cn/library/ms954603.aspx）。

今天，由于商业与法律的需要，例如美国的健康保险便利和义务法案（Health Insurance Portability and Accountability Act，HIPAA）——卫生保健组织机构很清楚要与它们的商业结合起来。遗憾的是，大多数的健康信息系统一直是私人所有，而且在一个卫生保健行业它们只为一个部门服务。

Health Level Seven（HL7）是美国国家标准化协会（ANSI）认可的标准化开发组织中的一个，它正在全世界保健行业里运行着（Level Seven 引用了开放系统互连模型 OSI 的最高层——应用层）。传统上，它从事临床建模与数据的管理工作，最近的一个版本——HL7 3.0 版本扩展到了各种卫生保健行业，如制药业、医疗设备及成像设备。

HL7 标准也指定了一些适当的信息基层组织，如 Web Services，它就适合传送 HL7 信息，并且在应用软件之间对于如何确保这个信息的传送的交互性，提供了一个说明性的向导。将 HL7 应用软件应用在 Web Services 上，意味着首先设计一个正确的体系结构，其次是提供一个可执行的满足 Web Services 的环境。本文只是涉及 HL7 Web Services Basic profile（HL7WSP）。

21.3.1 HL7 模型概念

通过对 HL7 标准规格说明书以及本文以外的一些工具的描述，这部分将介绍一些主要的 HL7 模型概念和人工制品，这些都与我们的讨论相关。

1．参考信息模型

对于一个给定的卫生保健领域，HL7 3.0 版本说明书是基于参考信息模型的（RIM）。这是一种公共的模型框架，包括病例模型、信息模型、交互模型、消息模型和实现信息说明书。

HL7 的参考信息模型是一个静态的卫生保健信息模型，它代表了至今为止负责 HL7 标准发展行为的卫生保健领域的各个方面。HL7 3.0 版本标准开发过程定义了一些规则，这些规则用于从参考信息模型中获取一些具体领域信息模型，从而在 HL7 规格说明书中使这些模型更精确，最后产生 XML 表单定义（XSD）与一个具体的消息类型联合起来。

2．消息结构

HL7 应用软件之间的交互行为是通过消息的交换来完成的。这样，在提供 envelopes 支持应用程序之间的消息交换期间，这个标准就提供了一个真实的功能水准。HL7 消息

的封装被称为 wrappers，最初是通过 RIM 中类的定义和关联模型化的。然后，这些说明书被用来为消息 wrappers 创建 XML 表单。接下来，在 HL7 消息开发框架中所列的过程在图 21-2 中有所描述。

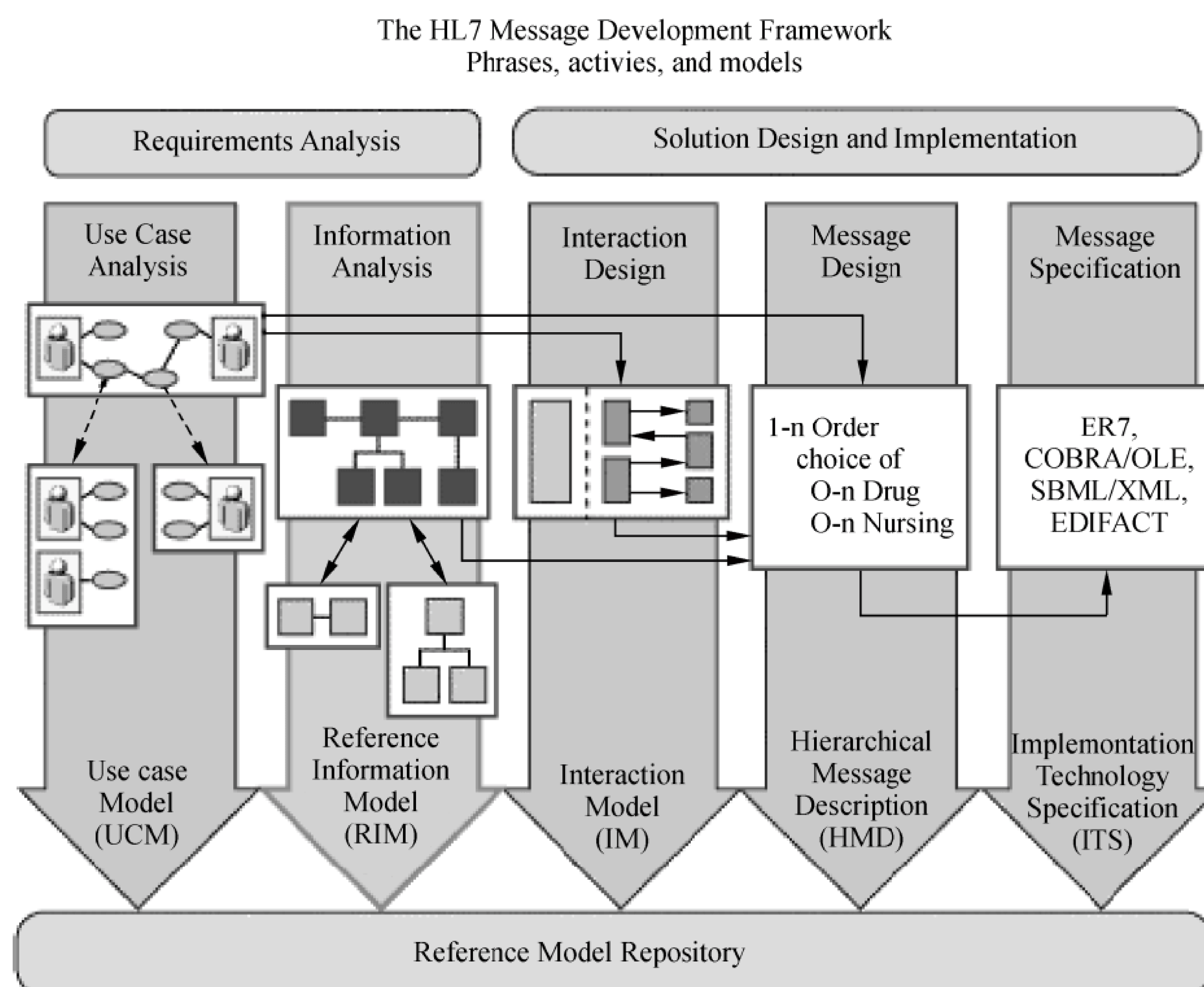


图 21-2　HL7 消息发展体系——引自 HL7 V3 Ballot 7

所有的 HL7 消息都被放在 Transmission Wrapper，Wrapper 的目的是支持应用软件之间消息的传输（和确认）。Wrapper 的重要部分是一些元素，如消息标志符、消息的创建时间、交互标志符、发送者和接收者标志符、确认编码和消息序列号（可选）。认为 HL7 消息是在合理的 HL7 应用软件之间进行交换这一点是很重要的。也就是说，特殊的软件应用或是组成成分（像“顺序实体”）都代表着有组织的或是可管理的实体（像西部医院登记一样）。所以，在传输层，发送者和接收者概念不会被看成是一个规格说明书的一部分。

3．交互

一次 HL7 交互就是信息特殊转移过程中的一次联合，一个触发事件就开始了消息的转移，应用软件进行接收和发送消息。在 HL7 里，一个触发事件是引起信息在应用软件之间进行转移的一系列精确条件，它也代表着一个真实的事件。例如，实验室顺序的安排或是一个病人的登记。

4．应用程序角色

HL7 里的每一个应用属于一个具体的应用程序角色。根据一个应用程序提供给其他应用程序的服务或是一个应用程序为了获得特定的服务而发送给其他应用程序的消息，这样一个角色就体现了应用程序的职责。

5．Storyboard

像消息类型、交互作用和应用程序角色这些概念都集合在了一个 HL7 Storyboard 里，它是用来指定在 HL7 标准化行为范围内与任意卫生保健领域相关联的用例。

一个 Storyboard 是由一小段记叙了它本身的目的及交互作用图表的描述所组成的（在应用层）应用程序角色间相互作用的级数。就像图 21-1 中的那样，交互作用的图表指明了相应交互作用的职责（就是应用程序角色）、交换信息的类型以及期望的信息交换的顺序。

21.3.2 体系结构

基于刚刚介绍的 HL7 概念模型，现在我们能更精确地定义出 HL7 应用。 这些都是在支持应用程序角色软件组成中的设计与实现，这些角色是作为交互行为中的一部分来实现发送者/接收者的职责，通过使用 Web 服务通信基层结构来满足 HL7 Web 服务的（如图 21-3 所示）。

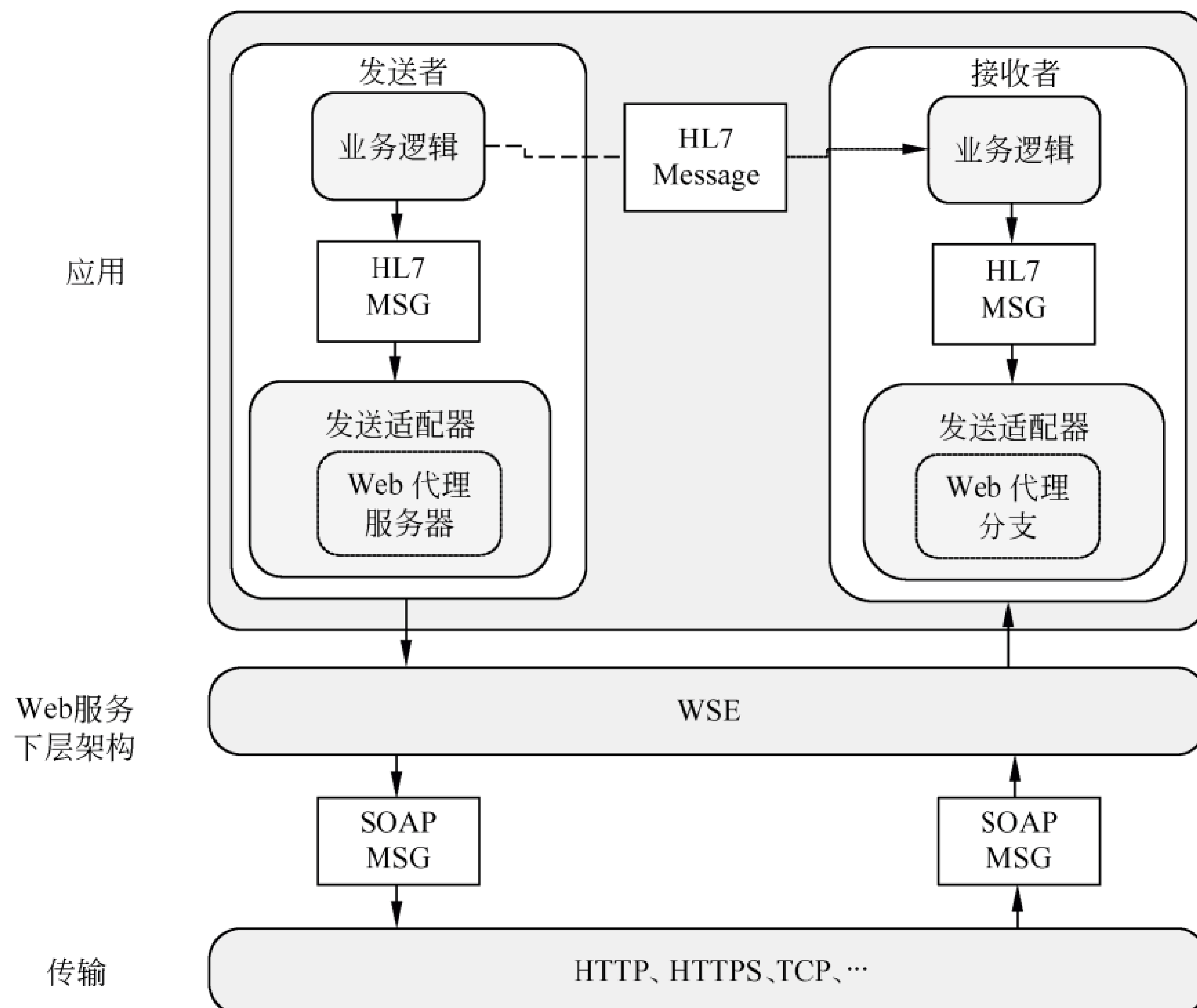


图 21-3 参考体系结构

在图 21-3 所示的结构里，能够抽象出 HL7 发送者/接收者内部的这两组功能：商业逻辑和 Web 服务适配器（需要强调的是，这里商业逻辑的范围是在 HL7 应用进行它们的发送者的角色和/或信息的接收者内部。也就是说，它支持一种具体的通信模式。应用层商业逻辑、消息的产生，或是为了响应需求而提供的具体服务这些都是在范围之外的）。

至于 HL7 消息的扩展，我们需要关注一下。商业逻辑的任务如下。

（1）发送端：创建一种具体 HL7 消息类型的 XML 描述，消息类型包含消息体、Transmission and Control Wrappers。将消息传送到 Web 服务适配器，适配器负责传送到接收应用端。

（2）接收端："找回"由 Web 服务适配器接收的 HL7 消息，同时从接收到的 XML 消息那里打开 Transmission Wrapper、Control Wrapper 和消息体；验证 HL7 消息是否满足用来交互的商业规则和约束；核实发送应用端是否需要一个应用层的确认信息（HL7 消息类型 MCI）——如果是那样的话，发送那个消息。

Web 服务适配器的功能主要是用来处理消息的分发和确认信息。因此，主要包括如下内容。

1）发送端

（1）读取接收到的 HL7 消息的 Transmission Wrapper，以便决定如何到达 Web 服务基层结构上的发送容器（例如接收应用软件），从而配置 SOAP。

（2）基于 HL7 消息类型、应用配置和规则（如安全性）来准备一个 SOAP 消息，包括作为一个 SOAP 消息体部分的 HL7 XML 消息，这个消息被发送到 Web 服务基层组织。

（3）把 SOAP 消息传递到 Web 服务代理，通过网络进行传输。

（4）无论发送端什么时候请求，都准备接收并存储来自接收端的相应信息或是应用层的确认消息。

2）接收端

（1）从 Web 服务站处接收 SOAP 消息。

（2）验证接收到的 SOAP 消息满足应用配置和一些约束条件（如安全性）。

（3）或者将这些接收到的消息在内存中以永久的形式保留。

（4）有选择性地从 SOAP 消息里打开 HL7 XML 消息，同时核对接收到的 HL7 消息是否与期望的 HL7 消息类型相符合。

（5）验证是否任意通信层的确认信息都需要被执行，在哪种情况下资金积累一个合适的消息发送到源消息发送端。

（6）传递 HL7 消息给接收应用端。

在适配器这层，这些情况都能够当作多个单行道方式或是请求/就答消息扩展模式来实现。在一个真正的实施过程中，适配器的结构也需要处理综合性应用和互操作能力。例如，如果一个应用业务逻辑不能直接与一个 Web 服务环境进行交互或是它被搭建在一个与以前实现时不同的平台上。

21.3.3 开发 HL7 Web 服务适配器

原则上，尤其是当范围被限制在只是支持 HL7 Web 服务时，开发 HL7 Web 服务就与开发普通的 Web 服务相类似了。事实上，RIM 的标准化模型的有效性，消息类型的说明书，通信模式及 Web 服务都在一定程序上影响着开发过程。为了高效地开发 HL7 Web 服务适配器，需要按如下步骤来做。

（1）消息和数据类型的设计。在一个像 HL7 这样面向消息的环境里开发一个 Web 服务，必须首先设计可交换的消息、已用的数据类型以及 XSD 表单里它们的说明书。这项活动完全受益于 HL7（使 XSD 表单自动化产生）所构造的消息和数据类型工具。

（2）适配器模式的选择。创建 Web 服务适配器的下一步是选择哪一个适配器结构模式能够最好地适合 HL7 通信模式，这个通信模式是由步骤（1）中所获得的消息类型来指定的。这一步要定义，比如说，一个（仅仅一个）代理/ Stub 组成成分是必要的。

（3）HL7 Web 服务契约开发。从一个普通的角度考虑，在创建一个面向消息的 Web 服务的下一步就能够定义它的契约了，用一种标准化的可用计算机处理的语言称作 Web 服务描述语言，或者在支持 Web 服务标准的编程语言里实现它的开发。

（4）产生 Web 服务 Stub 和代理的实现。一旦 WSDL 契约完成，它就可能创建使用一些工具的 Web 服务 Stub 和代理服务器，这些工具是由像 WSDL.exe 这样的开发平台所提供的。

（5）开发适配器业务逻辑。这一步是建立在前一步代码生成的基础上的，添加了必要的逻辑来支持适配器的功能，这些功能在 Architecture 一节里已描述过了。

一个普通的 WSDL 契约都详细说明了一个 Web 服务的名字和端口，通过这些端口，Web 服务器可以和客户端应用程序进行通信。一个端口指定了网络中服务生效的位置。每个端口也指定了端口上的一群有用的操作（portTypes），和客户与服务器在那个端口上进行通信的协议间的一个绑定。端口类型代表了暴露在 Web 服务上的各种接口。操作是接口的方法，它们定义了客户端请求服务端的输入信息，以及定义了服务器用于应答客户的输出信息。消息的格式也是基于 WSDL 契约中所定义的类型的格式（XML 表单）。

21.3.4 案例研究

一个参考实现案例已经构建了，包括两个系统之间的交互：医疗信息系统（Hospital Information System, HIS）和实验室信息系统（Laboratory Information System，LIS）。

（1）HIS 是由两个 Sub-systems 排序和报告组成的，为此应用程序和 Web 服务已经被开发。

（2）类似地，LIS 是由 Web 服务和业务逻辑组成的，Web 服务从 HIS 排序系统接收命令，业务逻辑是将确认信息返回到 HIS 排序或报告系统。

（3）这里，设想中用到的通信模式交换与前面所描述的“发送消息负载——附有确

认信息——立即”是相符的。

（4）为了保持业务逻辑的简单实施，当允许一些用户与样品应用程序进行交互时，两个 Windows 客户应用程序必须被开发。

HIS 客户应用程序发送命令请求给 HIS Web 服务器，并且显示发送命令的接收确认信息。它的用户界面允许用户发送一个命令（发送按钮），因为全球唯一的标识符(GUID)是由客户应用程序自动产生的。当 HIS 系统接收确认、信息确认和通信结果时，HIS 客户用户界面也会通过 LIS 系统（用三个验证框：OrderAck、ActiveConf 和 Result）显示出来。

下面是用来交换 HL7 信息的逐步流程，这些信息存在于提前设想的模板的上下文里。

（1）当用户接口从 HIS 客户机那里收到信号时，HIS 业务逻辑就会产生一个序号标识符，同时通过创建一个 XML 文件以及在 HL7 负载里加入一个序号 ID 来构造 POLB_IN2120 信息。

（2）业务逻辑发送一个 POLB_IN2120 信息（Send Order）给适配器，通过它的代理服务（POLB_AR002942 服务代理）来调用 LIS 服务。

（3）在 Laboratory 端，POLB_AR002942 Service Stub 接收到 SOAP 信息，同时使它对于 LIS Web 服务适配器是可用的。

（4）LIS 适配器从 SOAP 信息里得到 HL7 信息（Order），同时依据 HL7 信息类型表单来验证从 SOAP 那得到的被封装的 HL7 负载。

（5）LIS 适配器从 SOAP 信息里得到 HL7 信息（Order），同时依据 HL7 信息类型表单来验证从 SOAP 那得到的被封装的 HL7 负载。

（6）如果需要，它会准备确认序列，这个确认序列是通过构造一个 XML 文件同时在文件里附上一个预先定义的应答确认来实现的。

（7）当一个新的信息到达时，LIS 业务逻辑重新从顺序队列里得到 HL7 信息，并且将信息发送给 LIS 客户端。

事实上，对于给定的应用程序角色和交互活动，可以构造一个能自动产生代码的工具，用这个工具来创建需求信息队列和存储引入的信息。这是一种用来构建 Web 服务适配器代码的方法（代码案例见原文）。

21.3.5　结论

在卫生保健领域，HL7 是用来为协同工作而创建的基层结构。HL7 使用参考信息模型（RIM）来获得具体领域的信息模型，同时把它们精炼到 HL7 说明书中，结合具体的消息类型自动产生 XML 表单定义（XSD）。因为能够被设计所公用，因此这些概念就对它们进行建模，而不是只集中在关于互操作能力的一些技术问题上。我们能够考虑说明书，同时知道如何构建一个应用程序软件，包括角色、协作模式和消息。

从理论到实践，HL7 并没有告诉我们怎么构建和设计一些方案，而是当 Web 服务被用时，本文提到的参考体系结构就是一个相应的出发点。

21.4 以服务为中心的企业整合——案例分析

原文参见 URL（http://www.ibm.com/developerworks/cn/webservices/ws-soi2/）。

以一个经过简化的实际案例为例，介绍了以服务为中心的企业集成的基本步骤，从业务分析到服务建模，到架构设计，到系统开发的整个生命周期。以服务为中心的企业集成涉及到的主要技术被穿插在各个步骤中进行了详细的讲解。

21.4.1 案例背景

某航空公司的 IT 系统已有好几十年的历史。该航空公司的主要业务系统构建于 20 世纪七八十年代，以 IBM 的主机系统为主——包括运行于 TPF 上的订票系统和运行在 IMS 上的航班调度系统等。在这些核心系统周围也不乏基于 UNIX 的非核心作业系统，和基于.Net 的简单应用。这些形形色色的应用，有的用汇编或 COBOL 编写，运行于主机和 IMS 之上；有的以 PRO*C 编写，运行在 UNIX 和 Oracle 上。这些应用虽然以基于主机终端的界面，但是基于 Web 和 GUI 的应用也为数众多。

近年来，该公司在企业集成方面也是煞费苦心——已经在几个主要的核心系统之间构建了用于信息集成的信息 Hub（Information Hub），其他应用间也有不少点到点的集成。尽管这些企业集成技术在一定程度上增进了系统间的信息共享，但是面对如此异构的系统，技术人员依然觉得企业集成困难重重。

（1）因为大部分核心应用构建在主机之上，所以 Information Hub 是基于主机技术开发，很难被开放系统使用。

（2）Information Hub 对 Event 支持不强，被集成的系统间的事件以点到点流转为主，被集成系统间耦合性强。

（3）牵扯到多个系统间的业务协作以硬编码为主，将业务活动自动化的成本高，周期长，被开发的业务活动模块重用性差。

为了解决这些企业集成中的问题，该公司决定以 Ramp Control 系统为例探索一条以服务为中心的企业集成道路。本文将以 Ramp Control 系统中的 Ramp Coordination 流程为例，说明如何用以服务为中心的企业集成技术一步步解决该公司 IT 技术人员面临的企业集成问题。

21.4.2 业务环境分析

在航空业中，Ramp Coordination 是指飞机从降落到起飞过程中所需要进行的各种业务活动的协调过程，其流程图如图 21-4 所示。通常，每个航班都有一个人负责 Ramp

Coordination，这人通常称为 Ramp Coordinator。由 Ramp Coordinator 协调的业务活动有：检查机位环境是否安全、卸货、装货和补充燃料等。

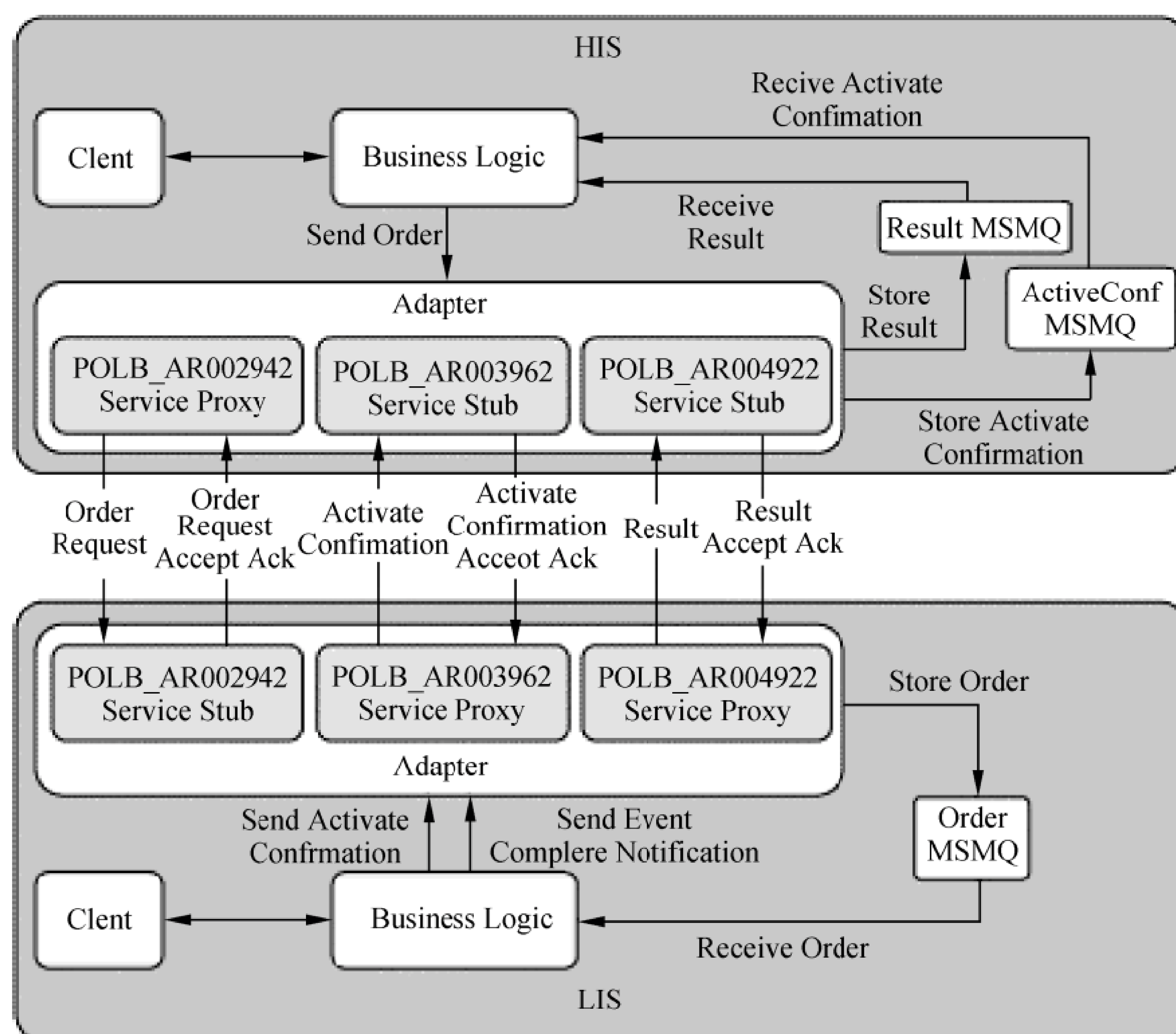


图 21-4　设想的体系结构的模板

实际上，Ramp Coordination 的流程因航班类型的不同，机型的不同有很大差异。图 21-5 所示的流程主要针对降落后不久就起飞的航班，这种类型的航班称为 short turn

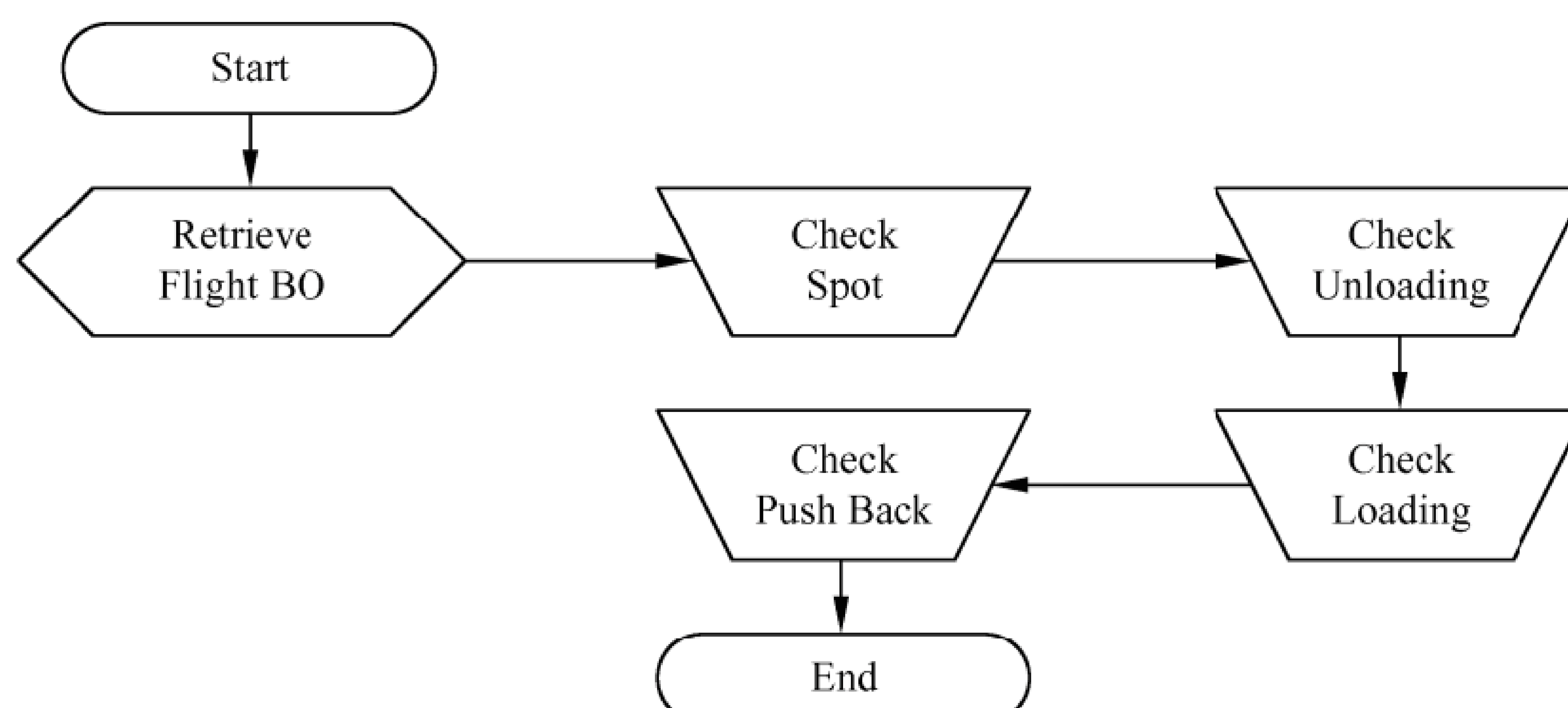


图 21-5　Ramp Coordination 流程图

around 航班。除了 short turn around 航班外，还有其他两种类型的航班，如图所示。Arrival Only 航班指降落后需要隔夜才起飞的，Departure Only 航班是指每天一早第一班飞机。这些航班的Ramp Coordination的流程和Short Turn Around类型的流程大部分的业务活动是相似的。这三种类型的航班根据长途/短途，国内/国外等因素还可以进一步细分。每种细分的航班类型的 Ramp Coordination 的流程都是略有不同。

很明显，如此多的流程之间共享着一个业务活动的集合，如此多种类型的流程都是这些业务活动的不同组装方式。以服务为中心的企业集成中流程服务就是通过将这些流程间共享的业务活动抽象为可重用的服务，并通过流程服务提供的流程编排的能力将它们组成各种大同小异的流程类型，来降低流程集成成本，加快流程集成开发效率的。以服务为中心的企业集成，通过服务建模过程发现这些可重用的服务，并通过流程模型将这些服务组装在一起。

服务建模

IBM 推荐使用组件业务建模（Component Business Model）和面向服务的建模和架构（Service-Oriented Model and Architecture）两种方法学建立业务的组件模型、服务模型和流程模型。

服务模型是服务建模的主要结果。Ramp Coordination 相关的服务模型及和 Ramp Coordination 流程相关的有两个业务组件，内容如下。

- Ramp Control：负责 Ramp Control 相关各种业务活动的组件。
- Flight Management：负责航班相关信息的管理，包括航班日程，乘客信息等。

这两个业务组件分别输出如下服务。

（1）Retrieve Flight BO：由 Flight Management 输出，主要用于提取和航班相关的数据信息。

（2）Ramp Coordination：由 Ramp Control 输出，主要用于 Ramp Coordination 流程的编排。

（3）Check Spot：由 Ramp Control 输出，用于检测机位安全信息。

（4）Check Unloading：由 Ramp Control 输出，用于检查卸货状况。

（5）Check Loading：由 Ramp Control 输出，用于检查装货状况。

（6）Check Push Back：由 Ramp Control 输出，用于检查关门动作。

在服务建模确定系统相关的服务输出后，还需要确定服务在当前环境下的实现方式。在我们的案例中，Retrieve Flight BO 被实现为信息服务，Ramp Coordination 被实现为流程服务，通过 BPEL4WS 方式实现。其他 4 个服务都是 Staff Service。需要注意的是，因为环境的不同和随着系统的演化，我们可能会改变服务的实现方式，如 Check Push Back 现在通过 Staff Service 即人工服务实现。将来随着自动化程度的增强，Check Push Back 完全可能通过自动化的系统实现。到那时，只需重新实现这个服务，而无需改变整个流程。这是服务的可替换性的一个典型实例。

21.4.3　IT 环境分析

在构建 Ramp Control 系统之前，该航空公司已经有大量的 IT 系统。作为架构设计的重要步骤的现有 IT 环境调研，描绘了和 Ramp Control 相关的 IT 系统的状况，包括周围应用和应用提供的接口，这些应用和 Ramp Control 交互的类型和数据格式。简化的 IT 环境视图，描绘了 Ramp Coordination 流程和周围系统交互状况。目前，Ramp Coordination 流程需要 4 种类型的外围应用交互。

（1）从乘务人员管理系统提取航班乘务员的信息。

（2）从订票系统中提取乘客信息。

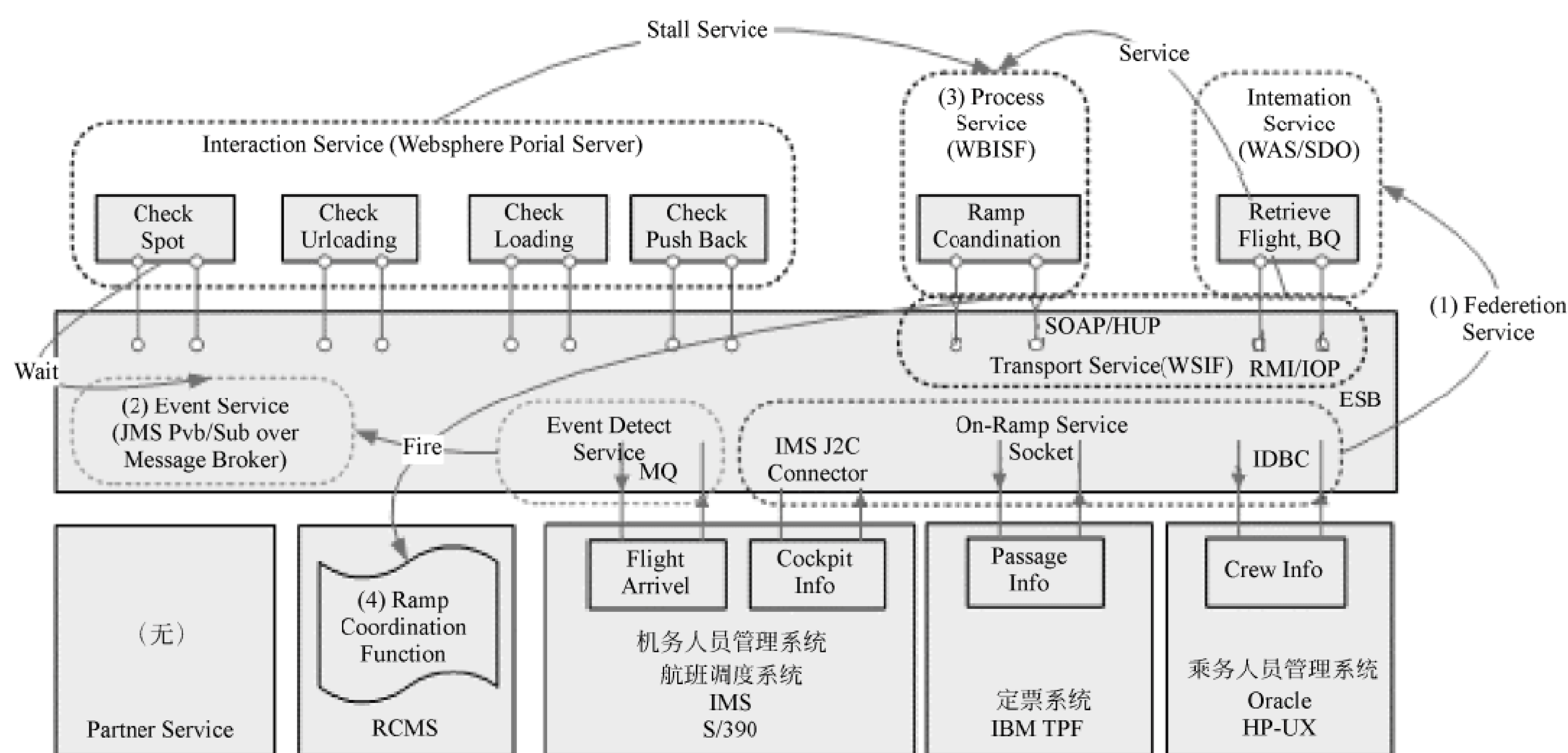
（3）从机务人员管理系统中提取机务人员信息。

（4）接收来自航班调度系统的航班到达事件。

通过将主机应用中的信息集中为粗粒度的业务对象，并通过信息服务输出，为该公司的核心系统提供了更加通用的连接能力，同时为 IT 系统的平滑演进提供了必需的条件。

21.4.4　高层架构设计

据需求和设计阶段的业务模型和现有 IT 环境调研结果，再结合传统的 IT 应用开发方法，Ramp Coordination 系统的高层架构被设计了出来，如图 21-6 所示。



错误!

图 21-6　Ramp Coordination 系统架构

如下 4 点简要介绍了本案例中的主要架构元素以及它们之间的工作关系。

（1）信息服务。Federation Service: Ramp Coordination 流程中需要从已有系统中提取 4 类信息，在 Service 建模阶段这 4 类信息被聚合为 Flight BO（Business Object）。如上文所述，Retrieve Flight BO 服务用于从已有系统中提取 Flight BO。它实际上是一个 Federation Service，将来自乘务人员管理系统、机务人员管理系统和订票系统中的信息聚合在一起。从这三个已有系统来的 Crew Info、Cockpit Info 和 Passage Info 是在已有系统中已经存在的业务逻辑或业务数据，它们属于可接入服务（on-ramp service），接入的协议分别为 JDBC、IMS J2C Connector 和 socket。乘务人员管理系统基于 Oracle 数据库，Crew Info 可以直接通过 JDBC 获得。机务人员管理系统基于 S/390 上的 IMS，IBM 已经提供了 IMS 的 J2C Connector，所以 Cockpit Info 可以通过 J2C connector 获得。订票系统构建在 IBM TPF 之上，由于实时性的要求，socket 是比较好的接入方法。Retrieve Flight BO 被实现为一个 EJB，外部访问通过 RMI/IIOP 绑定访问这个服务。在 Retrieve Flight BO 内部，Flight BO 以 SDO 来表示。

（2）企业服务总线中的事件服务。Event Service: 在检查机务环境安全（Check Spot）前，Ramp Coordiator 需要被通知航班已经到达。这个业务事件由航班调度系统激发，Flight Arrival 是典型事件发现服务（Event Detect Service），它通过 MQ 将事件传递给 Message Broker，通过 JMS 的 Pub/Sub，这个事件被分发给 Check Spot。这里的 Event Service 是本例中 ESB 的重要组成部分。通过 ESB 上的通用事件服务，现有 Information Hub 的缺陷得到了克服。应用程序间的事件集成不再需要点到点的方式，而是通过 ESB 的事件服务完成订阅发布，应用程序间的耦合性得到了极大的缓解。

（3）流程服务。Process Service: Ramp Coordination 被实现为一个 Process Service，它被 WBI SF 的 BPEL4WS 容器执行，BPEL4WS 容器提供 Choreograph Service、Transaction Service 和 Staff Service 支持。Ramp Coordination 通过 RMI/IIOP 协议调用，在 BPEL4WS 容器中 WSIF 被用于通过各种协议调用服务，它成为 ESB 中 Transport Service 的一部分。Ramp Coordination 中的人工动作被实现为 Staff Service 而集成到流程中。这里，Staff Service 通过 Portlet 实现，运行在 Websphere Portal Server 上。Portal Service 实现部分 Delivery Service 支持 PDA 设备，Ramp Coordinator 通过 PDA 设备访问系统。

（4）企业服务总线中的传输服务。RCMS 是即将新建系统，用于提供包括 Ramp Coordination 在内的 Ramp Control 的功能。RCMS 通过由 WSIF 实现的 Transport Service 以 SOAP/HTTP 调用 Ramp Coordination 服务。

21.4.5 结论

通过一个简单的案例，讲解了以服务为中心的企业集成的主要步骤和涉及的技术。这些集成的技术，无论是方法学，体系结构，还是编程模型都在不断的发展中。随着这些技术的不断完善，以服务为中心的企业集成方案的实施将更加简单高效。

附　录

和系统架构相关的一些国家标准如下。

标　准　号	标 准 名 称
GB 15843.2-1997	信息技术　安全技术　实体鉴别　第 2 部分：采用对称加密算法的机制
GB/T 16964.1-1997	信息技术　字型信息交换　第 1 部分：体系结构
GB/T 17191.1-1997	信息技术　具有 1.5Mb/s 数据传输率的数字存储媒体运动图像及其伴音的编码　第 1 部分：系统
GB/T 17191.2-1997	信息技术　具有 1.5Mb/s 数据传输率的数字存储媒体运动图像及其伴音的编码　第 2 部分：视频
GB/T 17191.3-1997	信息技术　具有 1.5Mb/s 数据传输率的数字存储媒体运动图像及其伴音的编码　第 3 部分：音频
GB/T 17235.1-1998	信息技术　连续色调静态图像的数字压缩及编码　第 1 部分：要求和指南
GB/T 17235.2-1998	信息技术　连续色调静态图像的数字压缩及编码　第 2 部分：一致性测试
GB/T 17975.2-2000	信息技术　运动图像及其伴音信号的通用编码　第 2 部分：视频
GB/T 17975.9-2000	信息技术　运动图像及其伴音信息的通用编码　第 9 部分：系统解码器的实时接口扩展
GB/T 18238.1-2000	信息技术　安全技术　散列函数　第 1 部分：概述
GB/T 18336.1-2001	信息技术　安全技术　信息技术安全性评估准则　第 1 部分：简介和一般模型
GB/T 18336.2-2001	信息技术　安全技术　信息技术安全性评估准则　第 2 部分：安全功能要求
GB/T 18336.3-2001	信息技术　安全技术　信息技术安全性评估准则　第 3 部分：安全保证要求
GB/T 18391.3-2001	信息技术　数据元的规范与标准化　第 3 部分：数据元的基本属性
GB/T 18391.4-2001	信息技术　数据元的规范与标准化　第 4 部分：数据定义的编写规则与指南
GB/T 18391.5-2001	信息技术　数据元的规范与标准化　第 5 部分：数据元的命名和标识原则
GB/T 18391.6-2001	信息技术　数据元的规范与标准化　第 6 部分：数据元的登记
GB/T 17975.3-2002	信息技术　运动图像及其伴音信号的通用编码　第 3 部分：音频
GB/T 17975.7-2002	信息技术　运动图像及其伴音信息的通用编码　第 7 部分：先进音频编码(AAC)
GB/T 16648-1996	信息技术　文本通信　标准页面描述语言（SPDL）
GB/T 17544-1998	信息技术　软件包　质量要求和测试
GB/T 17548-1998	信息技术　POSIX 依从性的测试方法
GB/T 17962-2000	信息技术　信息资源词典系统（IRDS）服务接口
GB/T 18221-2000	信息技术　程序设计语言、环境与系统软件接口　独立于语言的数据类型
GB/T 16681-2003	信息技术 开放系统中文界面规范

续表

GB/T 8567-1988	计算机软件产品开发文件编制指南
GB/T 9385-1988	计算机软件需求说明编制指南
GB/T 9386-1988	计算机软件测试文件编制规范
GB/T 11457-1995	软件工程术语
GB/T 12504-1990	计算机软件质量保证计划规范
GB/T 12505-1990	计算机软件配置管理计划规范
GB/T 13400.1-1992	网络计划技术　常用术语
GB/T 13400.2-1992	网络计划技术　网络图画法的一般规定
GB/T 13400.3-1992	网络计划技术在项目计划管理中应用的一般程序
GB/T 13502-1992	信息处理　程序构造及其表示的约定
GB/T 14079-1993	软件维护指南
GB/T 14246.1-1993	信息技术　可移植操作系统界面　第一部分：系统应用程序界面（POSIX.1）
GB/T 14394-1993	计算机软件可靠性和可维护性管理
GB/T 15532-1995	计算机软件单元测试
GB/T 15538-1995	软件工程标准分类法
GB/T 15853-1995	软件支持环境
GB/T 15936.4-1996	信息处理　文本与办公系统　办公文件体系结构（ODA）和交换格式　第四部分：文件轮廓
GB/T 16260-1996	信息技术　软件产品评价　质量特性及其使用指南
GB/T 16647-1996	信息技术　信息资源词典系统（IRDS）框架
GB/T 16680-1996	软件文档管理指南
GB/T 16682.1-1996	信息技术　国际标准化轮廓的框架和分类方法　第 1 部分：框架
GB/T 16682.2-1996	信息技术　国际标准化轮廓的框架和分类方法　第 2 部分：OSI 轮廓用的原则和分类方法
GB/T 16684-1996	信息技术　信息交换用数据描述文卷规范
GB/T 17917-1999	商场管理信息系统基本功能要求
GB/T 18234-2000	信息技术　CASE 工具的评价与选择指南
GB/T 8566-2001	信息技术　软件生存周期过程
GB/T 18491.1-2001	信息技术　软件测量　功能规模测量　第 1 部分：概念定义
GB/T 18492-2001	信息技术　系统及软件完整性级别
GB/Z 18493-2001	信息技术　软件生存周期过程指南
GB/T 18714.1-2002	信息技术　开放分布式处理　参考模型　第 1 部分：概述
GB/T 18714.2-2002	信息技术　开放分布式处理　参考模型　第 2 部分：基本概念
GB/T 18905.1-2002	软件工程　产品评价　第 1 部分：概述
GB/T 18905.2-2002	软件工程　产品评价　第 2 部分：策划和管理
GB/T 18905.3-2002	软件工程　产品评价　第 3 部分：开发者用的过程
GB/T 18905.4-2002	软件工程　产品评价　第 4 部分：需方用的过程

续表

GB/T 18905.5-2002	软件工程　产品评价　第 5 部分：评价者用的过程
GB/T 18905.6-2002	软件工程　产品评价　第 6 部分：评价模块的文档编制
GB/Z 18914-2002	信息技术　软件工程　CASE 工具的采用指南
GB/T 18714.3-2003	信息技术 开放分布式处理 参考模型 第 3 部分：体系结构
GB/T 3453-1994	数据通信基本型控制规程
GB/T 12453-1990	信息处理系统　开放系统互连　运输服务定义
GB/T 12500-1990	信息处理系统　开放系统互连　面向连接的运输协议规范
GB/T 15127-1994	信息处理系统　数据通信　双扭线多点互连
GB/T 15128-1994	信息处理系统　开放系统互连　面向连接的基本会话服务定义
GB/T 15129-1994	信息处理系统　开放系统互连　服务约定
GB/T 16262-1996	信息处理系统　开放系统互连　抽象语法记法——（ASN.1）规范
GB/T 16263-1996	信息处理系统　开放系统互连　抽象语法记法——（ASN.1）基本编码规则规范
GB/T 16264.1-1996	信息技术　开放系统互连　目录　第 1 部分：概念、模型和服务的概述
GB/T 16264.2-1996	信息技术　开放系统互连　目录　第 2 部分：模型
GB/T 16264.3-1996	信息技术　开放系统互连　目录　第 3 部分：抽象服务定义
GB/T 16264.4-1996	信息技术　开放系统互连　目录　第 4 部分：分布式操作规程
GB/T 16264.5-1996	信息技术　开放系统互连　目录　第 5 部分：协议规范
GB/T 16264.6-1996	信息技术　开放系统互连　目录　第 6 部分：选择属性类型
GB/T 16264.7-1996	信息技术　开放系统互连　目录　第 7 部分：选择客体类
GB/T 16264.8-1996	信息技术　开放系统互连　目录　第 8 部分：鉴别框架
GB/T 16644-1996	信息技术　开放系统互连　公共管理信息服务定义
GB/T 16645.1-1996	信息技术　开放系统互连　公共管理信息协议　第 1 部分：规范
GB/T 16724.3-1997	信息技术　系统间的远程通信和信息交换 X.25 DTE 一致性测试　第 3 部分：分组层一致性测试套
GB/T 15629.4-1997	信息处理系统　局域网　第 4 部分：令牌传递总线访问方法和物理层规范
GB/T 17178.1-1997	信息技术　开放系统互连　一致性测试方法和框架　第 1 部分：基本概念
GB/T 9387.1-1998	信息技术　开放系统互连　基本参考模型　第 1 部分：基本模型
GB/T 17533.1-1998	信息技术　开放系统互连　远程数据库访问　第 1 部分：类属模型、服务与协议
GB/T 17533.2-1998	信息技术　开放系统互连　远程数据库访问　第 2 部分：SQL 专门化
GB/T 17969.1-2000	信息技术　开放系统互连　OSI 登记机构的操作规程　第 1 部分：一般规程
GB/T 17969.5-2000	信息技术　开放系统互连　OSI 登记机构的操作规程　第 5 部分：VT 控制客体定义的登记表
GB/Z 15629.1-2000	信息技术　系统间远程通信和信息交换　局域网和城域网　特定要求　第 1 部分：局域网标准综述
GB/T 17969.6-2000	信息技术　开放系统互连　OSI 登记机构的操作规程　第 6 部分：应用进程和应用实体

续表

GB/T 9387.2-1995	信息处理系统　开放系统互连　基本参考模型　第 2 部分：安全体系结构
GB/T 9387.3-1995	信息处理系统　开放系统互连　基本参考模型　第 3 部分：命名与编址
GB/T 9387.4-1996	信息处理系统　开放系统互连　基本参考模型　第 4 部分：管理框架
GB/T 15695-1995	信息处理系统　开放系统互连　面向连接的表示服务定义
GB/T 15696-1995	信息处理系统　开放系统互连　面向连接的表示协议规范
GB/T 16505.1-1996	信息处理系统　开放系统互连　文卷传送、访问和管理　第 1 部分：概论
GB/T 16505.2-1996	信息处理系统　开放系统互连　文卷传送、访问和管理　第 2 部分：虚文卷存储器定义
GB/T 16505.3-1996	信息处理系统　开放系统互连　文卷传送、访问和管理　第 3 部分：文卷服务定义
GB/T 16505.4-1996	信息处理系统　开放系统互连　文卷传送、访问和管理　第 4 部分：文卷协议规范
GB/T 16505.5-1996	信息处理系统　开放系统互连　文卷传送、访问和管理　第 5 部分：协议实现一致性声明形式表
GB/T 16646-1996	信息技术　开放系统互连　局域网媒体访问控制（MAC）服务定义
GB/T 16652-1996	开放文件体系结构（ODA）和互换格式　文件结构
GB/T 16687-1996	信息处理系统　开放系统互连　联系控制服务元素协议规范
GB/T 16688-1996	信息处理系统　开放系统互连　联系控制服务元素服务定义
GB/T 16967-1997	信息技术　开放系统互连　作业传送和操纵基本类及完全协议规范
GB/T 17965-2000	信息技术　开放系统互连　高层安全模型
GB/T 17967-2000	信息技术　开放系统互连　基本参考模型　OSI 服务定义约定
GB/T 17973-2000	信息技术　系统间远程通信和信息交换　在因特网传输控制协议（TCP）之上使用 OSI 应用
GB/T 18237.1-2000	信息技术　开放系统互连　通用高层安全　第 1 部分：概述、模型和记法
GB/T 18237.2-2000	信息技术　开放系统互连　通用高层安全　第 2 部分：安全交换服务元素（SESE）服务定义
GB/T 18237.3-2000	信息技术　开放系统互连　通用高层安全　第 3 部分：安全交换服务元素（SESE）协议规范
GB/T 18794.1-2002	信息技术　开放系统互连　开放系统安全框架　第 1 部分：概述
GB/T 18794.2-2002	信息技术　开放系统互连　开放系统安全框架　第 2 部分：鉴别框架
GB/T 18237.4-2003	信息技术　开放系统互连　通用高层安全　第 4 部分：　保护传送语法规范
GB/T 18794.3-2003	信息技术 开放系统互连 开放系统安全框架 第 3 部分：访问控制框架
GB/T 18794.4-2003	信息技术 开放系统互连 开放系统安全框架 第 4 部分：抗抵赖框架
GB/T 18794.5-2003	信息技术 开放系统互连 开放系统安全框架 第 5 部分：机密性框架
GB/T 18794.6-2003	信息技术 开放系统互连 开放系统安全框架 第 6 部分：完整性框架
GB/T 18794.7-2003	信息技术 开放系统互连 开放系统安全框架 第 7 部分：安全审计和报警框架